Real-time scalable video coding for surveillance applications on embedded architectures

Real-time Scalable Video Coding for Surveillance Applications on Embedded Architectures

Real-time Scalable Video Coding for Surveillance Applications on Embedded Architectures

PROEFONTWERP

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op maandag 15 december 2014 om 16:00 uur

door

Marijn Johannes Hendricus Loomans

geboren te Waalre

De documentatie van het proefontwerp is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr.ir. A.C.P.M. Backx
1e promotor:	prof.dr.ir. P.H.N. de With
copromotor(en):	dr.ir. E.G.T. Jaspers (ViNotion B.V.)
leden:	prof.dr.ir. M. van der Schaar (University of California Los Angeles)
	prof.dr.ir. R.L. Lagendijk (Delft University of Technology)
	prof.dr. J.J. Lukkien
	prof.dr.ir. G. de Haan
adviseur(s):	ing. C.J. Koeleman (ViNotion B.V.)

To my wife, family, Sumi and Mojo†

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Marijn Loomans

Real-time Scalable Video Coding for Surveillance Applications on Embedded Architectures / by Marijn Loomans. - Eindhoven : Technische Universiteit Eindhoven, 2014.
A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-3748-8
NUR 959
Subject headings: video compression / scalable video coding / wavelets / embedded systems / real-time systems / video surveillance

Cover drawing: M.C. Escher's "Print Gallery" © 2014 The M.C. Escher Company B.V. - Baarn - the Netherlands. All rights reserved. www.mcescher.com
Cover photograph: M.J.H. Loomans - "Grünerløkka little planet, Oslo" © 2014
Cover design: M.J.H. Loomans

Summary

Video surveillance systems are rapidly evolving from analog to digital systems, but this transition encounters several bottlenecks. Furthermore, a large diversity is emerging in: (1) the scale of video surveillance systems, from a few cameras in a small office to hundreds of cameras in a public area, and (2) the differentiation in computing power between decoding systems, from hand-held devices to central video-surveillance rooms. These aspects and the system requirements of professional surveillance hampers the straightforward use of broadly accepted video compression standards. In practice, a surveillance system is subject to serious design and cost constraints, hence it should typically be a resource-constrained system with limited cost.

The above-mentioned aspects and requirements motivate the work described in this thesis, which aims at the design of a specific video coding system for surveillance. For this purpose, we investigate a flexible scalable video coding strategy that avoids recoding of video signals, while its complexity enables mapping on resource-constrained systems. Three sub-systems are investigated: scalable image coding, scalable temporal coding and motion estimation.

Prior to addressing these three sub-systems, Chapter 2 introduces the reader to various aspects of Scalable Video Coding. We concentrate on wavelet-based scalable coding, first by introducing common wavelet transforms and algorithms from 1D to multi-level 2D, after which the discussion moves to two state-of-the art wavelet coefficient encoding algorithms. More specifically, we present SPIHT (Set Partitioning in Hierarchical Trees) and SPECK (Set Partition Embedded bloCK), and provide a detailed example for each, illustrating their inner working.

Chapter 3 has a clear focus on the algorithms for encoding of wavelet coefficients. First, the complexity of the SPIHT and SPECK coding algorithms is investigated, after which we propose a hardware-efficient scalable image coder based on SPECK, called TSSP (Two-Stage SPECK), which creates an output stream identical to SPECK. We propose several enhancements that significantly improve the algorithm efficiency, to facilitate embedded implementations. To control the com-

plexity, we split the coefficient processing into a data-independent stage and a data-dependent stage. The algorithm eliminates the need for dynamic lists, and processing in the second stage is performed for all bit planes in parallel. The algorithm is enhanced with high scalability, allowing parsing of the bitstream without payload decoding. This enables the creation of a bitstream of any desired quality, resolution and order, which is attractive for surveillance.

Chapter 4 explores the trade-off between complexity and performance in scalable wavelet coders in the temporal domain. We present a framework to evaluate various temporal configurations of the coding system and adopt a special configuration (BDLD) suited for video surveillance applications, featuring low memory access and low end-to-end delay, while still achieving a sufficiently high quality. We also introduce two extensions to the framework. The first improves the energy correction in the temporal lifting tree and significantly reduces computational complexity and the associated memory bandwidth. The second extension improves the average quality of the frames within a GOP and significantly reduces quality fluctuations, which is also expressed by a 30% reduction of the PSNR variance, while clearly improving the perceptual quality. As a result, the coding quality approaches that of H.264 SVC within 1 dB.

In Chapter 5 we present a novel motion estimator, designed specifically for scalable video coding, which is based on hardware acceleration features. The proposed Highly Parallel Predictive Search (HPPS) algorithm features two candidate vectors and supplements these positions with additional refinement candidates arranged in a Parallelogram-Shaped Scanning (PSS) pattern. The PSS pattern reduces SAD test points up to 50% compared a full search around the two candidate vectors, without reducing the accuracy of the found motion vectors. HPPS also features hierarchical, multi-level candidate prediction, so that large motion vectors can still be found. Compared to other motion estimators, the computational load of HPPS is fixed, regardless of scene activity and temporal distance, thereby ensuring an efficient mapping on computing cores.

Chapter 6 describes a complete implementation of our scalable video codec system consisting of optimized implementations of two-dimensional wavelet filtering, the TSSP wavelet coefficient encoder and the temporal filtering framework. For a custom-designed surveillance camera and video encoder, we first investigate possible target architectures for its embedded computing platform. Then based on cost, size and power consumption requirements, a programmable DSP is adopted. Using this platform, we achieve efficient implementations by using SIMD (Single Instruction Multiple Data) instructions for data and computational parallelism. Furthermore, we exploit Direct Memory Access (DMA) to transfer data to and from our self-managed Level-2 cache, parallel to computations. The platform ob-

tains an execution time of 6.08 ms to perform a 4-level wavelet transform at 4CIF resolution and 75 TSSP encodings per second. Finally, we integrate the full temporal filtering, but without motion estimation and compensation, due to architectural limitations. The fully scalable video encoding system executes at 20 fps.

The proposed algorithms and their mapping and execution on an embedded DSP architecture have shown that it is possible to specifically enhance algorithms during the design phase for future mapping on a target architecture. We can therefore conclude that scalable coding is feasible for video surveillance systems, and that enhanced algorithms can even be executed on a resource-constrained embedded system. Despite the feasibility, the large-scale introduction of the above coding techniques is hampered by the large technical variation in architectures and implementations at customer premises, often leading to customized solutions per manufacturer.

Samenvatting

Videobewakingssystemen ontwikkelen zich in rap tempo van analoge naar digitale systemen, maar deze overgang brengt verschillende complicaties aan het licht. In de praktijk heeft een videobewakingssysteem te maken met diverse systeembeperkingen, zoals een beperkte rekenkracht door vermogensconsumptie en een gangbare gelimiteerde kostprijs van bepaalde componenten. Videobewakingssystemen hebben een grote diversiteit in de volgende aspecten: (1) de schaal van het systeem, die kan variëren tussen een paar camera's in een klein kantoor tot honderden camera's voor publieke ruimtes, en (2) de differentiatie in rekenkracht tussen verschillende decodeersystemen, variërend van draagbare systemen tot centrale ruimtes voor videobewaking met speciale apparatuur. Tevens wijken de eisen aan de videocompressie in een professioneel videobewakingssysteem af van de eisen voor consumentenproducten. De voorgaande aspecten bemoeilijken het rechtstreeks toepassen van reeds gestandaardiseerde videocompressietechnieken.

Dit proefschrift gaat over het ontwerpen van een speciaal videocompressiesysteem voor videobewaking, waarbij zoveel mogelijk de bovengenoemde beperkingen worden gedisconteerd. Dit speciaal ontworpen videocompressiesysteem heeft als eigenschap dat de video maar eenmaal gecomprimeerd hoeft te worden, waarna een videosignaal met afwijkende resolutie, kwaliteit en aantal beelden per seconden hiervan kan worden afgeleid (schaalbaarheid). Tevens is dit compressiesysteem zodanig ontworpen dat een efficiënte afbeelding mogelijk is op een hardwarearchitectuur met beperkte middelen, zoals rekenkracht, geheugen en koeling. De volgende drie deelsystemen worden onderzocht: schaalbare codering van afbeeldingen, schaalbare codering van video, en bewegingsschatting binnen de videocodering. Tenslotte worden de nieuw ontworpen deelsystemen samengevoegd in een experimenteel ontwerp, dat wordt getest op zowel beeldkwaliteit als compressieperformance.

Hoofdstuk 2 geeft een overzicht van verschillende facetten van schaalbare videocodering, waarbij de focus ligt op videocompressie gebaseerd op wavelets. Eerst worden verschillende wavelet transformaties en compressietechnieken be-

sproken, gevolgd door een- en twee-dimensionale transformaties met verschillende representaties van de beeldinformatie met betrekking tot resolutie. Hierna worden twee geavanceerde compressiealgoritmes specifiek voor wavelets besproken: SPIHT (Set Partitioning in Hierarchical Trees) en SPECK (Set Partition Embedded bloCK). Voor deze algoritmes wordt een gedetailleerd voorbeeld gegeven dat de manier van berekenen illustreert.

Hoofdstuk 3 beschouwt de algoritmes voor het coderen van waveletcoëfficiënten. Allereerst wordt de complexiteit van de SPIHT en SPECK algoritmes vergeleken, waarna een schaalbaar videocompressiealgoritme wordt voorgesteld dat efficiënt afgebeeld kan worden op een architectuur met specifiek voor multimedia geschikte hardwarecomponenten. Dit algoritme produceert een gecomprimeerde informatiestroom identiek aan SPECK, en wordt TSSP (Two-Stage SPECK) genoemd. Daarbij worden verschillende optimalisaties gepresenteerd, die de schaalbaarheid van het compressiesysteem en de implementatie op een hardwarearchitectuur met beperkte rekenkracht verbeteren. Hiervoor scheidt het algoritme gegevensafhankelijke berekeningen van gegevensonafhankelijke berekeningen in twee stappen. Daarnaast worden dynamische lijsten geëlimineerd en berekeningen in de tweede stap worden gelijktijdig uitgevoerd voor alle bitlagen. De schaalbaarheid wordt verbeterd zodat het gecodeerde videosignaal eenvoudig kan worden getransformeerd naar een signaal met andere coderingsparameters, zonder dit signaal te decoderen. Deze transformatie kan een videosignaal creëren met variaties in kwaliteit, resolutie en volgorde van gegevens en dit maakt TSSP zeer aantrekkelijk voor toepassing in een videobewakingssysteem.

Hoofdstuk 4 onderzoekt de afweging tussen complexiteit en kwaliteit in schaalbare compressiesystemen voor video. Het hoofdstuk start met een kader om verschillende temporele structuren te evalueren, hetgeen uiteindelijk leidt tot een speciale structuur (BDLD) die geschikt is voor videobewakingssystemen door zijn lage vertraging, beperkte complexiteit en goede beeldkwaliteit. Tevens worden er twee uitbreidingen gepresenteerd voor het compressiesysteem. De eerste uitbreiding verbetert de temporele energiebalans tussen de verschillende beeldrepresentaties door het reduceren van de complexiteit en de benodigde geheugenbandbreedte. De tweede uitbreiding verbetert de gemiddelde kwaliteit en reduceert fluctuaties van de kwaliteit binnen een periodieke groep van videobeelden. Deze verbetering is meetbaar in de vorm van een meer constante kwaliteit (30% minder fluctuaties) en ook als een duidelijke visuele verbetering. Met beide uitbreidingen benadert het voorgestelde videosysteem de kwaliteit van de H.264-SVC standaard binnen 1 dB.

In Hoofdstuk 5 wordt een nieuwe bewegingsschatter gepresenteerd, die specifiek ontworpen is voor schaalbare videocompressie en een efficiënte afbeelding

mogelijk maakt op systemen met hardwarematige versnelling van berekeningen. Deze bewegingsschatter wordt Highly Parallel Predictive Search (HPPS) genoemd en gebruikt twee kandidaatvectoren die elk omringt worden door aanvullende locaties geplaatst in een patroon gevormd naar een parallellogram (PSS). Dit patroon reduceert het aantal SAD berekeningen met 50% vergeleken met een normaal vierkant zoekgebied, zonder verlies van de nauwkeurigheid van de gevonden bewegingsvectoren. HPPS heeft ook een vectorpropagatie tussen verschillende temporele beeldrepresentaties, waardoor ook grote bewegingen gevonden kunnen worden. In tegenstelling tot vele andere bewegingsschatters is de complexiteit van HPPS constant en onafhankelijk van de inhoud van de video en de afstand tussen de beelden. Dit heeft als gevolg dat HPPS zeer geschikt is voor een real-time implementatie.

Hoofdstuk 6 presenteert de implementatie van het schaalbare compressiesysteem uit de voorgaande hoofdstukken en beschrijft geoptimaliseerde implementaties van de twee-dimensionale wavelet transformatie, het TSSP compressiealgoritme en de temporele filterstructuur. Verschillende hardwarearchitecturen worden onderzocht voor toepassing in een speciaal voor videobewaking ontworpen camera. Een programmeerbare DSP is gekozen gebaseerd op kostprijs, opgenomen vermogen en benodigde ruimte. Parallellisatie van berekeningen en gegevens is bereikt door veelvuldig gebruik te maken van SIMD (Single Instruction Multiple Data). Daarnaast is Direct Memory Access (DMA) gebruikt voor het gelijktijdig berekenen en verplaatsen van gegevens naar de Level-2 cache, zodat de cache toegang volledig handmatig wordt beheerd. Voor 4CIF resolutie wordt de wavelet transformatie met 4 lagen berekend in 6.08 ms en voor TSSP kunnen 75 coderingsiteraties per seconde worden uitgevoerd. Wanneer ook de volledige temporele transformatie wordt toegepast (zonder bewegingsschatter, vanwege beperkingen van de hardwarearchitectuur), codeert het volledig schaalbare systeem 20 beelden per seconde.

De algoritmebeschrijvingen gecombineerd met de implementatie en de metingen op een DSP architectuur hebben aangetoond dat het mogelijk is om algoritmes voor schaalbare videocodering zodanig te ontwerpen dat een zeer efficiënte afbeelding kan worden gerealiseerd op een architectuur met specifiek voor multimedia geschikte hardwarecomponenten. Dit heeft geleid tot een innovatief ontwerp van een speciaal ontworpen videobewakingscamera. Een directe grootschalige introductie van de beschreven technieken wordt echter afgeremd door de grote variatie van systemen in videobewaking die reeds in gebruik zijn bij klanten, met specifieke implementaties per producent.

Contents

List of abbreviations

Abbreviations defined in this thesis

3DSB 3-Dimensional SubBand coding

53EC 5/3 Energy Correction

BDLD BiDirectional Low Delay

BP Bit Plane

BPR Bit-Plane Reduction

EC Energy Correction

HPPS Highly Parallel Predictive Search

HS Highly Scalable

LCEF Low-Complexity Encoder Feedback

PSS Parallelogram-Shaped Scanning

SL Significance Level

TSSP Two-Stage SPECK

TEC Temporal Energy Correction

TQC Temporal Quality Consistency

QRF Quality-Reduced Frame

Other abbreviations used in this thesis

3DRS 3-Dimensional Recursive Search

4CIF 4× CIF: resolution of 704 × 576 pixels

AD Analog-to-Digital (Conversion)

ALU Arithmetic Logic Unit

ARPS Adaptive Rood Pattern Search

ASIC Application-Specific Integrated Circuit

ATM Automated Teller Machine

BME Block-based Motion Estimation

CABAC Context-Adaptive Binary Arithmetic Coding

CAVLC Context-Adaptive Variable-Length Coding

CCD Charge-Coupled Device

CMOS Complementary MetalOxide Semiconductor

CCTV Closed-Circuit TeleVision

CIF Common Intermediate Format: resolution of 352 × 288 pixels

CGS Course-Grained Scalability

CODWT Complete-to-Overcomplete Discrete Wavelet Transform

DCT Discrete Cosine Transform

DDR Double Data Rate

DMA Direct Memory Access

DWT Discrete Wavelet Transform

DSP Digital Signal Processor

DVB Digital Video Broadcasting (With -C, -T or -S suffix for Cable, Terrestrial and Satellite, respectively)

EBCOT Embedded Block Coding with Optimal Truncation

EZW Embedded Zerotree Wavelet

EPZS Enhanced Predictive Zonal Search

FFT Fast Fourier Transform

FGS Fine-Grained Scalability

FIR Finite Impulse Response

FPGA Field Programmable Field Array

FPS Frames Per Second

FS Full Search

GOP Group Of Pictures

HD High-Definition: Broadcast resolution standard of 1280×720 pixels or 1920×1080 pixels

HVSBM Hierarchical Variable Size Block Matching

IIR Infinite Impulse Response

LAN Local Area Network

LIP List of Insignificant Pixels (used by SPIHT)

LIS List of Insignificant Sets (used by SPIHT)

LSB Least-Significant Bit

LSP List of Significant Pixels (used by SPIHT)

LZC Listless Zerotree Coding

MAC Multiply-ACcumulate

MC Motion Compensation Motion Compensator

MCTF Motion Compensated Temporal Filtering

ME Motion Estimation Motion Estimator

MSB Most-Significant Bit

MSE Mean Square Error

NLS No List SPIHT

PMVFAST Predictive Motion Vector Field Adaptive Search Technique

PSNR Peak Signal-to-Noise Ratio

QCIF Quarter CIF: resolution of 176×144 pixels

QOS Quality-Of-Service

QR Quality-Rate

RD Rate-Distortion

RDC Rate-Distortion-Complexity

RQ Research Question

RTOS Real-Time Operating System

RVLC Reversible Variable Length Code

SD Standard-Definition: Broadcast resolution of 704×576 pixels (PAL) or 704×480 pixels (NTSC)

SPIHT Set Partitioning in Hierarchical Trees

SPECK Set Partition Embedded bloCK

SIMD Single Instruction Multiple Data

SNR Signal-to-Noise Ratio

SVC Scalable Video Coding

UHD Ultra High-Definition: Broadcast resolution proposal of 3840×2160 pixels

UMCTF Unconstrained Motion Compensated Temporal Filtering

VBSMC Variable Block-Size Motion Compensation

VCR Video Cassette Recorder

VLC Variable Length Code/Coding

VLIW Very Long Instruction Word

VLSI Very Large Scale Integration

CHAPTER 

Introduction

All truths are easy to understand once they are discovered, the point is to discover them.

GALILEO GALILEI

Abstract

This thesis begins with a brief history of video surveillance, the migration from analog to digital systems, and presents the components of a modern video surveillance system. Then the focus is on video surveillance systems, where it is motivated why their requirements and observers differ from consumer video distribution systems, and what design constraints a practical surveillance system has to satisfy. From this we derive requirements for the coding system, which allows us to construct a detailed problem statement for this thesis. The outline of the individual chapters is presented in a structured manner to guide the reader throughout this thesis, and for each chapter the contributions and their related publications are listed.

1.1 Brief history of video surveillance

Emergence of video surveillance

With the advent of all-electronic video camera technology based on cathode ray tubes in the 1940s, the basis was created for CCTV (Closed-Circuit TeleVision). One of the first known CCTV systems was installed in Germany by Siemens, to observe the launch of V2 rockets in 1942. However, only when the VCR (Video Cassette Recorder) was introduced in the 1970s, CCTV became more widespread, as events could be recorded and reviewed at later times, without the need of continuous human observation. Since then, the usage of CCTV has expanded strongly, so that such systems have arrived at a status where numerous cameras are deployed. For example, according to the BBC in 2009 [1], in London alone, almost 7,500 surveillance cameras are operated by the authorities. This excludes surveillance cameras operated by private businesses, such as those inside shops and for example the surveillance cameras present in and around Automated Teller Machines (ATMs). When taking into account all surveillance cameras, both indoor and outdoor, a 2011 study [2] estimates the total number of surveillance cameras in the UK to be around 1.8 million, which translates to one camera for every 32 UK citizens. While the UK is known to be heavily monitored by CCTV, it does give an indication of the prevalence of surveillance cameras in our daily lives. Although the social aspects of video surveillance are a very interesting and important topic, the rest of this thesis will focus on the technical aspects of these systems.

Migration to digital recording systems

Until fairly recently, video surveillance systems were predominantly analog sytems. Images were initially captured by a cathode ray tube, and later through CCD (Charge-Coupled Device) and CMOS (Complementary MetalOxide Semiconductor) sensors, which convert light to electrical signals. This electric signal is transferred along an extensive cable network to a central surveillance room, where the video is displayed on monitors and/or recorded on magnetic tapes by a VCR, usually one for each camera. These systems require frequent changing of tapes, and an equal amount of machines in the field (the cameras) and in the central surveillance room (the VCRs). As a result, these systems required large investments in infrastructure and considerable operational costs due to staffing and maintenance.

With the continuous growth of processing power and storage capabilities of personal computers, digitization of analog video signals has become a viable option. The handling of digital video signals became feasible when video compression was gradually introduced, to reduce the recording bandwidth of digital video

signals. To put the amount of data that a single video camera produces in perspective, a digitized color video signal at standard-definition broadcast resolution generates enough data to fill one CD-ROM every 22 seconds. To handle this vast amount of data, digital recording systems utilize two data reducing strategies. The first involves reducing the number of frames stored per second, from 25 per second contained in the original signal to, for example, one frame per second, thereby effectively reducing the amount of video data by a factor of 25. Secondly, each frame is compressed using advanced image coding algorithms, to remove visual redundancies, typically capable of reducing the amount of data by a factor of 20 without a significant loss in visual quality. Both strategies combined reduce the amount of data by a factor of 500, allowing a single CD-ROM to store 3 hours of video surveillance. Most of the video surveillance systems in use today still utilize these strategies.

Recently, due to the prevalence of use of H.264[1] on the Internet and Blu-ray Disc, systems are released that utilize H.264 video compression. Without frame-rate decimation, H.264 is capable of reducing the data by a factor 50-60, retaining much better motion resolution. However, due to the complexity of this codec, encoding and decoding requires significant resources, so that it is usually performed by a dedicated hardware chip, capable of coding a single high-quality video stream. Furthermore, due to the dependency on previously decoded frames, severe artifacts can occur when a small portion of the video stream is lost, or when coding deadlines are not satisfied. These characteristics have hampered the speed of introduction of H.264 in large-scale surveillance applications, despite its excellent compression ratio.

Migration to digital distribution networks

With the rapid growth of the Internet, IP-based (Internet Protocol) networks have grown in popularity. A similar popularity can be witnessed in CCTV applications, where the traditional analog connections have been replaced by extensive Local Area Networks (LANs) dedicated for video surveillance. The change of analog distribution to digital distribution has also significantly changed the composition of the video surveillance system, shifting the Analog-to-Digital (AD) conversion and compression of the video signal from the central surveillance location, to each individual camera. Since video signals are now transported digitally, the quality loss over large distances, common for analog systems, is avoided. Furthermore, due to the networked technology, a dedicated cable to the central location is not

[1] In the MPEG community, this standard is also known as MPEG-4 AVC.

required anymore, thereby simplifying the infrastructure and greatly improving the extensibility of the system.

1.2 Components of modern video surveillance systems

Figure 1.1 gives an overview of the components typically present in a modern video surveillance system. At the left side, we observe a large amount of networked cameras, both static and motion controlled. These cameras are connected through several network switches to a dedicated LAN, shielded from other networked applications for security reasons. To this dedicated network various observers are connected, each with their own characteristics. Storage servers record the video data on large sets of hard disks, allowing video information to be retained for days, weeks or months, depending on customer requirements and local legislation. Usually, a central surveillance location is an integral part of the infrastructure, to offer joint control and overview of all captured streams using multiple monitors. Due to the large amount of video streams, typically an overview is given of all the video streams in small windows, while the video streams that are currently of interest, are enlarged to full screen. At other locations in the building, a single observation monitor can be present, for example at a reception desk, to specifically observe the entrance. Mobile devices with a range of screen sizes and processing power may be connected wirelessly, such as laptops, tablets and smartphones. Finally, the surveillance system can be connected to the outside world by means of a modem or a specific and protected network connection, so that videos can be observed from remote locations. Recently, the list of observers has been expanded with automated video processing systems, that utilize advanced machine learning techniques to gain understanding of the behavior of objects in the video stream. For example, these systems can perform face and license plate recognition, tracking of persons or objects and analysis of the behavior of people, such as flow of crowds or posture estimation. A recent elaborate coverage of the use of automated systems in object categorization and detection in the domain of video surveillance can be found in Wijnhoven's thesis [3].

From Figure 1.1, we can derive different use cases for a video coding system, leading to the following key observations.

- A video surveillance system has a large amount of video capturing devices.
- The video capturing devices are usually small, low-powered systems, and can be enhanced with embedded system technology.

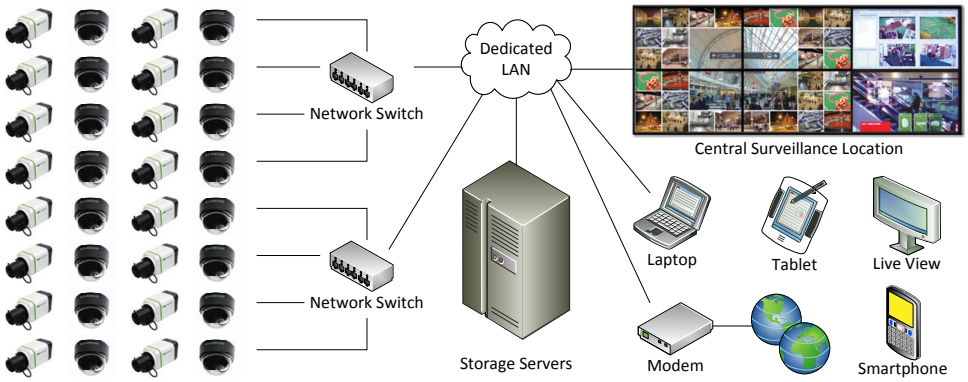


Figure 1.1: Components in a modern video surveillance system.

- A large network is present that connects all video capturing devices, storage servers and observers together, with limited bandwidth along connections.
- It is possible that a large amount of video information flows to a single location, such as a storage server, or a central surveillance location.
- Usually there is a limited amount of observers, but there is a rather large diversity between them. This diversity ranges from smartphones with limited processing power, to central surveillance locations with a large amount of screens and processing power, to enormous storage servers that allow storage of the captured video of hundreds of cameras for multiple years.
- Observers are not necessarily human, there is a fast growing segment of computers, processing and analyzing the video information autonomously.

We can foresee that these key observations will have an impact on the design of the video coding in a video surveillance system. First of all, we observe a differentiation of requirements when compared to traditional consumer video distribution systems. The number of video sources and observers is different, and special observer patterns are present, such as trick play, and non-human autonomous systems. Second, some form of scalability should be included due to the large differences between observers. Finally, due to the high number of low-power, low-cost systems and convergence of large amounts of data at a single location, complexity is of utmost importance. These impacts on video coding form the basis for the research scope of this thesis, which is discussed in more detail in the next section.

1.3 Research scope

Up to this point, key observations for the domain of video surveillance have been formulated from a broad system perspective. This section explores these key observations in more detail and distillates the impact of the domain constraints on the video coding algorithms and architecture within the video surveillance system. In the previous section, we have identified three areas of interest: (1) the different requirements of video surveillance systems compared to consumer video distribution systems, (2) a large differentiation between observers, requiring some form of scalability in the video coding, and (3) the need for controlled complexity, to facilitate low-cost, low-powered devices and data convergence at a single location. These areas of interest will be explored in more detail in the following sections.

Requirement differentiation for video surveillance systems

Compared to consumer video distribution systems, video surveillance systems have very different properties. A few of these can be directly derived from the key observations, of which a selection of the most critical properties are collected in Table 1.1.

Table 1.1: Differentiation of requirements between consumer video distribution and video surveillance systems. Number are not exact, but indicate orders of magnitude.

Aspect	Consumer distribution	Video surveillance
Number of simultaneous sources	1-2	1-1000
Number of observers	Millions	1-10
Diversity of observers	Standardized	Very high and dynamic
Allowed encoding complexity	Very high	Very limited
Allowed decoder complexity	Very low	Moderate
Simultaneous sources observed	1-2	1-1000
Camera control	No	Yes
Latency constraints	10 seconds (live) to none	1/5th of a second
Special playback modes	Limited importance	High importance

Because of the above differentiation of requirements, consumer video encoding standards are not optimal, as they have been defined mostly for the case of inexpensive, large-scale video distribution, not video surveillance. The following conclusions can be drawn if we take into account the special requirements of video surveillance systems.

1. *Encoder complexity* is very important because of the high number of video sources in the system and the cost-constrained nature of the video capturing devices.
2. *A powerful single observer and control room* should be able to decode tens to hundreds of streams simultaneously, where some quality degradation is acceptable, as long as it can be controlled.
3. *Live viewing* with optional camera control of the camera occurs frequently; therefore, end-to-end delay is a critical parameter.
4. The system should allow for *integrated special playback modes* such as slow motion and backwards play for analysis purposes and high-speed playback for searching key events.

Observer variation: novel observers and event dependencies

As mentioned previously, not all observers are human. Video analysis plays an increasingly important role in modern systems. Moreover, the use of visual signals in analysis depends on the application. For example, a high-quality image at a high frame rate is required for tracking, posture analysis and fall detection. Some applications investigate only the contents of the video on a per image basis, e.g. face recognition, and require only a single high-quality image at a fixed interval. Other applications investigate motion patterns and require only reduced image quality, but at a high frame rate. In this category falls for example the estimation of traffic-flow patterns in traffic video for congestion detection or detection of a ghost driver. In his work related to healthcare video processing, Albers [4] concludes that analysis video processing is characterized by occasional interrupts, only requesting information when needed, which results in much more variable access patterns when compared to traditional stream-oriented video processing.

The quality of the video that is required for a certain task can also be dependent on events, such as turning on a PTZ (Pan Tilt Zoom) camera. Alternatively, a desire for modified quality can also occur due to a change of scene events. For example, the resolution and frame rate for surveilling an empty car park can be relatively low, but when objects start moving around, a high quality and frame rate is desired. This adaptive requirement of quality does not occur only on live video, but is also present for stored video. Recent video capturings demand a high quality in case that analysis of the video is required, but archived video may be reduced in quality in order to save storage space.

From this observer variation, we can conclude there is a strong requirement for flexible access with respect to three system parameters: quality, resolution and

frame rate of the same video data. This leads to the exploration of three degrees of scalability, based on these three system parameters. The degree of scalability can be optimized specifically for the observer of the video, for both live and recorded video. To ensure a fluent use between such forms of scalability, this scalability should be realized without any recoding, but by simply discarding parts of the coded information.

Design constraints for practical surveillance systems

For a practical video surveillance system, the use of resource-constrained systems is unavoidable. The following items lead to constraints for the video coding system.

- The video cameras are distributed over a certain premises. Due to the high number of cameras, their cost should be reasonable and size should be small, while power consumption / heat dissipation should be limited.
- The global cost and size constraint on the applied video cameras in a system limits the amount of intelligence that can be incorporated inside.
- Data storage servers and the control room should be able to cope with the large amount of video data that converges at these locations, which is particularly critical when this information is decoded on site.
- Display of these large amounts of data in the control room should be possible. However, reduced-scale display on a per camera basis is allowed, as long as the scale can be adjusted in an instance. This option should also be available for recorded video.
- Observation of the scene by PTZ cameras should be nearly instant, to allow human operators fine control of the camera movements.
- The video coding algorithms should allow these flexible modes of observation, without resorting to computationally expensive recoding of video information.

Derived video coding system requirements

In the previous sections, we have highlighted several system aspects that define the scope of the research in this thesis. We have first discussed the different requirements for video surveillance systems compared to consumer video systems in Section 1.3, followed by a explanation on the variation of observers, both novel

observers and event dependencies in Section 1.3. Finally, in Section 1.3, we have presented the design constraints that a practical surveillance system imposes to the system architect. From these sections, we can distill three critical system requirements for a video coding system applicable to video surveillance, which are as follows.

1. The most critical is the need for *scalability* in the video codec. This is required for all of the above items: different video coding requirements for video surveillance (Section 1.3), observer differentiation (Section 1.3) and the capability for processing large amounts of converging video streams by a single observer without recoding (Section 1.3).

2. Furthermore, the *complexity* of the video coding is of utmost importance, and has a special focus on encoder complexity. This is due to the high number of encoding sources, limited in cost and processing ability (Section 1.3). Similarly, the decoding complexity is important for observers that need to handle large amounts of streams converging at a single location (Section 1.3).

3. The video coding should be *specifically designed for the video surveillance domain* and incorporate limitations on end-to-end delay and feature a feasible implementation of special playback modes (Sections 1.3 and 1.3).

1.4 Problem statement and research questions

Definition of problem statement

The requirements formulated at the end of the previous section have been used to delineate the problem statement for this thesis, which is summarized as follows:

To design a video coding system suited for video surveillance, featuring flexible scalable video coding that avoids recoding of video signals, while its complexity enables mapping on resource-constrained embedded systems.

This problem statement embodies three critical aspects: (1) the main objective, (2) the feature requirements and (3) the domain constraints. In the following, we will elaborate on these three critical aspects.

Objective *To design a video coding system suited for video surveillance.* Preceding sections have clearly illustrated the specific nature of video surveillance systems and their requirements. Due to the high number of encoders, the complexity of the encoding algorithm is very important. Furthermore, we have to

consider that all videos should be potentially processed by a single powerful observer, yet still with finite processing power and a broad variety of devices.

Scalability and trick-play requirements *Flexible scalable video coding that avoids recoding of video signals.* A flexible scalable video system will facilitate playback by any type of observer by providing access to various qualities, resolutions or frame rates that the observer requires or can handle. The system should also provide video surveillance specific features, such as trick play and mechanisms for gradually erasing earlier recordings. Furthermore, the features should be obtained without video signals, to minimize the complexity.

Domain constraints *While its complexity enables mapping on resource-constrained embedded systems.* Most of the video capturing devices in the video surveillance system will be cameras distributed over a certain premises. Due to the large amount of cameras in a typical surveillance system, they should be of reasonable cost, limited size, and should not require large amounts of power or forced cooling. Most video capturing devices are therefore likely to be resource-constrained embedded systems, which should be considered from the first design onwards.

Research questions

Since the objective, scalability and trick-play requirements and domain constraints are defined, it is possible to formulate four clusters for our research questions: (1) a high-performance coefficient coding algorithm is required to encode scalable transformed image and video data, (2) we need an framework for and complexity estimation of scalable temporal video coding, (3) the requirement of adapting motion estimation to scalable video coding and (4) the coefficient and temporal coding need to be mapped and verified on embedded architectures. These clusters lead to the following Research Questions (RQs) for this thesis.

RQ1: Efficient Variable Length Coding (VLC) of scalable transformed image and video data.

RQ1a: *Is it possible to design an alternative high-performance scalable video coefficient coding algorithm without losing coding fidelity?*

RQ1b: *Can we extend beyond the scalable properties of current scalable codecs, and provide highly flexible coding, with only a limited increase of codec complexity?*

RQ1c: *Is it better to implement scalable variable length coding in a quality-progressive way, or to process multiple qualities in parallel?*

RQ2: Framework for and complexity estimation of scalable temporal video coding.

RQ2a: *Can we identify a suitable framework for scalable temporal coding?*
RQ2b: *What is the trade-off between temporal coding complexity, end-to-end coding delay and corresponding visual quality?*
RQ2c: *Based on the outcomes of the previous trade-offs, is the chosen temporal coding structure suited for video surveillance?*

RQ3: Motion estimation in scalable video coding and its implementation efficiency.

RQ3a: *Can we develop a motion estimation algorithm that is suitable for a temporal scalable wavelet coding system and that elegantly propagates motion information within such a system?*
RQ3b: *Is it possible to design a motion estimation algorithm that is not only efficient, but also based on the hardware media processing kernels of modern computing platforms?*

RQ4: Verification of scalable VLC and scalable temporal coding on embedded architectures.

RQ4a: *What embedded execution architecture would be suitable for implementing a scalable video coding algorithm?*
RQ4b: *How can a combination of the high-performance scalable video coefficient coding and the scalable temporal coding be mapped on a resource-constrained embedded architecture, and can real-time performance be obtained?*

1.5 Contributions of this thesis

The contributions of this thesis are fully related to the design of a wavelet transformation based video codec, featuring motion estimation and compensation, and advanced wavelet coefficient coding. The details of the coding stages of such a coding system are explained in Chapter 2. Here we summarize the scientific and other technical contributions that were made in those coding stages.

Hardware-efficient encoding of wavelet coefficients

Our first contribution involves the re-design of one of the best wavelet coefficient encoding algorithms, called SPECK. The SPECK algorithm is a variant of the well-known SPIHT algorithm, but more efficient in terms of exploiting local coefficient correlation. We have designed a new SPECK-based coding algorithm that splits the algorithm in two processing stages: one data-independent stage and one data-dependent stage. This new concept has the advantage that it facilitates data-independent processing, which enables a significant improvement in deploying optimization techniques such as parallel computing and efficient memory management by avoiding the expensive use of coefficient memory lists. Furthermore, we propose a novel coded-coefficient data partitioning that groups coefficients at resolution and bit-plane levels, so that the scalability of the bitstream for data visualization is significantly increased. This new partitioning enabled flexible scalability without recoding, or a need to decode the payload, with a negligible increase in complexity or decrease of coding efficiency. Besides the above key innovations, we contribute a few other efficiency improvements. First, pixels are processed for all bit planes in parallel, instead of a per bit-plane strategy, thereby significantly increasing the processing speed. Second, we propose an enhancement for the definition of energy bands to use the lossless 5/3 integer wavelet, to enable efficient coding in a range from lossy to lossless.

Complexity analysis of scalable wavelet coding systems for video

We contribute with a flexible scalable coding framework based on four basic building blocks based on wavelet lifting configurations, which facilitate the creation of various temporal configurations in a flexible way. This contribution enables easy complexity analysis and control in the temporal domain. To this end, we investigate the computational complexity of various temporal configurations in terms of motion estimation complexity, memory usage and end-to-end delay. As a result of this broad analysis, we propose a special temporal configuration for video surveillance, which features good coding capabilities, low end-to-end delay and manageable complexity.

Architecture-driven motion estimation algorithm

As a consequence of using a special temporal configuration, we have developed a novel motion estimator with the following contributions. First, the algorithm is designed in such a way that it utilizes hardware-accelerated block-difference calculators and block-shifter units as fundamental operations, to smoothly fit with

programmable media processing kernels. Here, an attractive additional feature is that the algorithm has a fixed processing time for all cases. Second, motion-vector propagation of the motion estimator is matched to the hierarchical representation by propagating motion vectors from low-resolution wavelet levels to high-resolution wavelet levels in the temporal direction. Third, we evaluate this motion-vector propagation strategy for both the proposed motion estimation algorithm and a non-scalable state-of-the-art motion estimation algorithm.

Verification of designs on a resource-constrained embedded system in real-time

The above innovations and algorithms have been verified by executing them on a selected commonly available embedded system platform. These implementation studies have resulted in a number of efficiency improvements. First, we have found a highly efficient multi-level 2D integer wavelet transform implementation, utilizing SIMD parallelization and highly efficient background memory transfers using DMA to offload the CPU. These techniques are also exploited for a highly efficient implementation of the hardware-efficient encoding of wavelet coefficients. Furthermore, we validate the new temporal configuration of the scalable coding system with an evaluation of the practical implementation. Together, the previous points offer a prototype of the scalable video coding system, executing in real-time on a programmable state-of-the-art digital signal processor.

1.6 Thesis outline and related publications

This section gives an overview of the contents of this thesis and visualizes the relation between the chapters by means of Figure 1.2. In this figure, the topics are organized in the horizontal direction, and conceptual levels in the vertical direction. Chapter 2 provides an introduction to scalable wavelet video coding and introduces the four main concepts of scalable video coding: the wavelet transform, scalable spatial coding, scalable temporal coding and motion estimation. The main contributions of this thesis are presented in Chapters 3, 4 and 5, following the structure of Chapter 2. The validation and related efficiency improvements are presented in Chapter 6, while the conclusions and future work are discussed in Chapter 7. The individual chapters and their related scientific background are now briefly presented in more detail.

Chapter 2 introduces the reader to various aspects of Scalable Video Coding. It starts with motivating the use of video compression and gives a short history of current image and video coding. Then the focus is on wavelet-based

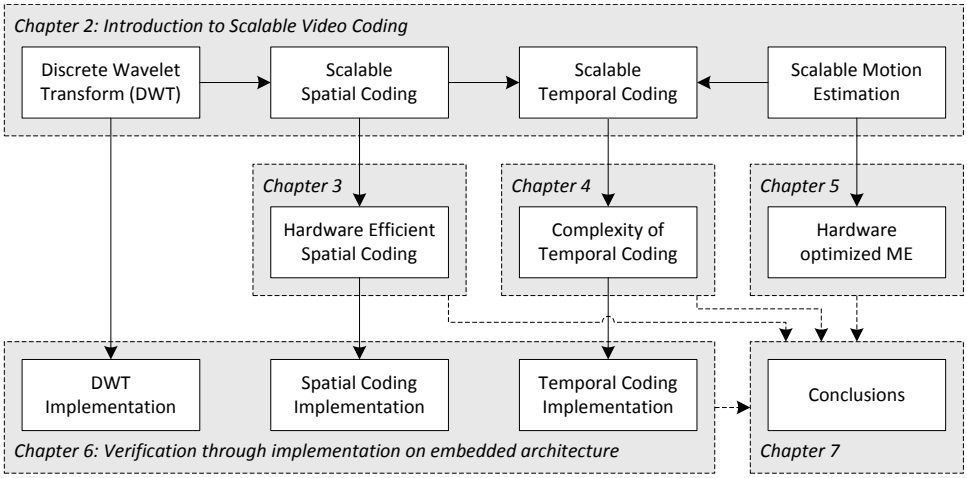


Figure 1.2: Research scope of this thesis, following the main structure of a scalable video codec in horizontal direction and different conceptual levels in vertical direction.

scalable coding, first by introducing common wavelet transforms and algorithms from 1D to multi-level 2D, after which two state-of-the art wavelet coefficient encoding algorithms are discussed. More specifically, we present SPIHT (Set Partitioning in Hierarchical Trees) and SPECK (Set Partition Embedded bloCK) and provide a detailed example for each, illustrating their inner working.

Chapter 3 has a clear focus on the algorithms for encoding wavelet coefficients, starting with image coding. We first investigate the complexity of the SPIHT and SPECK coding algorithms, after which we propose a hardware-efficient scalable image coder based on SPECK, called TSSP (Two-Stage SPECK), which creates an output stream identical to SPECK. We propose several enhancements that significantly improve the algorithm efficiency, to facilitate embedded system implementations. First, we split processing into one data-independent stage and one data-dependent stage. A highly efficient buffer eliminates the need for dynamic lists, while processing in the second stage is performed for all bit planes in parallel. An enhancement of the algorithm is the Highly Scalable (HS) extension, which allows parsing of the bitstream without payload decoding, thereby creating a bitstream of any desired quality, resolution and bitstream order. The second enhancement, 5/3 Energy Correction (53EC), retains the perfect reconstruction feature of the 5/3 integer wavelet, while significantly improving lossy coding performance.

The contributions of this chapter have been presented at the ICCE 2011 [5] and ICIP 2013 [6] conferences and further published in the IEEE Transactions on Consumer Electronics in 2011 [7].

Chapter 4 explores the trade-off between complexity and performance in scalable wavelet coders in the temporal domain. The chapter presents a framework to evaluate various temporal configurations of the coding system. We adopt a special configuration for video surveillance applications, featuring low memory access and low end-to-end delay, while still achieving sufficiently high quality. We also introduce two extensions to the framework. The first improves energy correction in the temporal lifting tree and significantly reduces computational complexity and the required memory bandwidth. The second improves the average quality of the frames within a GOP and reduces quality fluctuations significantly, which is also expressed by a reduction of variance of the PSNR by 30%. The second extension also significantly improves the perceptual quality. In all, the coding quality approaches that of H.264 SVC within 1 dB.

Contributions from this chapter have been presented at the WIC 2007 [8] and VCIP 2008 [9] conferences with respect to the temporal complexity estimation. The proposed enhancements have been presented at the PCS 2010 [9].

Chapter 5 presents a novel motion estimator designed specifically for scalable video coding, which is based on hardware-acceleration features. The proposed Highly Parallel Predictive Search (HPPS) algorithm features two candidate motion vectors and supplements these positions with additional refinement candidates arranged in a Parallelogram-Shaped Scanning (PSS) pattern. The PSS pattern reduces SAD test points up to 50% without significantly reducing the accuracy of the found motion vectors. HPPS also features hierarchical, multi-level candidate prediction so that large motion vectors can still be found. In contrast with many other motion estimators, the computational load of HPPS is fixed, regardless of scene activity and temporal distance, thereby ensuring an efficient mapping on processing cores.

The HPPS motion estimation algorithm, and enhancement to its predictions were presented at the ICIP 2009 [10] and ICIP 2010 [11], respectively.

Chapter 6 presents optimized implementations of two-dimensional wavelet filtering, the TSSP wavelet coefficient encoder and the temporal filtering framework. The chapter describes a complete implementation of the proposed scalable video codec system using a common embedded DSP platform. We

achieve efficient implementations by using SIMD (Single Instruction Multiple Data) instructions for data and computational parallelism, while exploiting Direct Memory Access (DMA) to transfer data to and from a self-managed Level-2 cache, parallel to computations. We are able to execute a 4-level wavelet transform at 4CIF (CCIR-601) broadcast resolution in 6.08 ms, while for TSSP, a real-time performance is obtained. Finally, the full temporal filtering is integrated, but without motion estimation and compensation, due to architectural limitations. The implementation of the fully scalable video encoding system obtains a frame rate of 20 fps.

The wavelet implementation was presented at the DSP 2009 conference [12], the TSSP codec implementation at the VCIP 2011 conference [13], and the combined video system at both the ICME 2009 [14] and VCIP 2011 [13] conferences.

Chapter 7. The final chapter in this thesis summarizes the achieved results and provides conclusions per chapter. Then the research questions of this thesis are addressed. We conclude that scalable coding is feasible for video surveillance systems and that it can be executed on resource-constrained embedded systems. The chapter concludes with a view beyond this thesis, and suggests areas of interest for future research, such as object-based video coding, combined with video analysis.

CHAPTER 

Developments in wavelet image and video coding

All this time the guard was looking at her, first through a telescope, then through a microscope, and then through an opera glass.

Through the Looking Glass
LEWIS CARROLL

Abstract

This chapter introduces the reader to various aspects of Scalable Video Coding. It commences with explaining the need for video compression and presents a short history of current image and video coding. From there, the attention shifts to wavelet-based scalable video coding, first by introducing common wavelet transforms and algorithms from 1D to multi-level 2D. After this, we discuss in more detail two state-of-the art wavelet coefficient encoding algorithms: SPIHT (Set Partitioning in Hierarchical Trees) and SPECK (Set Partition Embedded bloCK). For each coding technique, a detailed example is provided, which illustrates their operation.

2.1 Introduction

The first chapter has defined the scope of the thesis by discussing a brief history of video surveillance, the migration from analog to digital systems, and presenting the components of a modern video surveillance system. It has been clarified that the requirements of video surveillance systems differ from consumer video distribution systems and that a practical surveillance system has design constraints, from which requirements for the coding system can be derived.

This chapter provides a background for the reader in the area of video coding, with an emphasis on wavelet-based scalable coding. The organization of this chapter is visualized by the block diagram of Figure 2.1. In the first section (Block A) coding systems are discussed, while in the following sections the structure of scalable wavelet video coding is followed, as depicted in Block B in the figure. The four blocks in Block B indicate the relevant areas regarding various parts of this structure: the wavelet transform, coefficient coding, scalable video coding and motion estimation. The first two areas (Blocks C and D) are of introductory nature and will be discussed in dedicated sections in this chapter. The remaining blocks in Block B, on scalable temporal coding and motion estimation, and Block D provide the input to the contributing chapters of this thesis.

In Section 2.2 coding systems are discussed, presenting the two mostly used pseudo-frequency-domain transforms for image and video coding: the DCT and wavelet transformation. It also provides a brief discussion on the H.264 standard, which is the most prominent compression standard at this moment. Section 2.3 concentrates on the wavelet transform and discusses the lifting framework, multilevel wavelets, energy correction for both 1D and 2D wavelets and integer wavelet transforms. Section 2.3 presents two state-of-the-art wavelet coefficient coding techniques: SPIHT (Set Partitioning in Hierarchical Trees) and SPECK (Set Partition Embedded bloCK). The section also illustrates the operation of these coding techniques with examples. The discussion of these two coding techniques provide a good primer for the contributions presented in the next chapter on hardware-efficient scalable video coding, where we improve the SPECK codec.

2.2 Coding systems

The need for compression

Natural images captured by still and video cameras contain a large amount of digital information. Multiple Bytes are used per pixel to represent full-color information, while pixel resolutions are in the order of a million pixels per image.

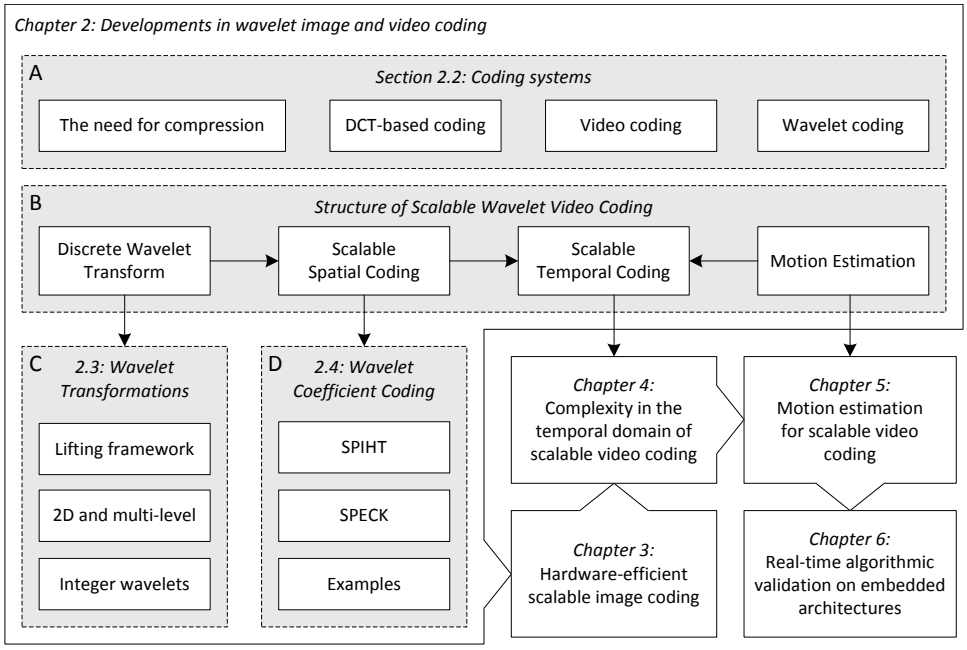


Figure 2.1: Layout of Chapter 2, based on the scalable wavelet video coding structure with relations to relevant techniques described in this chapter and upcoming chapters.

For video systems, tens of these images are captured per second, thereby creating a continuous stream of data in the order of 10-300 MegaBytes per second. Two common video formats in use today are based on SD (Standard-Definition) and HD (High-Definition) resolution. SD video has a spatial resolution of 720×576 pixels, with a refresh rate of 25 frames per second in Europe, while HD video has a resolution of 1920×1080 pixels, with refresh rates of 24, 25, 30 and even 60 frames per second. For compression, both formats typically use a color space with one luminance component (Y) and two color components ($C_r C_b$), which are commonly sub-sampled by a factor of two in both the horizontal and vertical dimension. This color space and sub-sampling is denoted in abbreviated form by the term 4:2:0.

Table 2.1 presents the amount of data generated by video systems at various spatio-temporal resolutions. Several common SD and HD formats are listed, together with the raw and target data rates. These rates lead to a desired compression ratio and associated storage requirements for 1 hour of video. It can be noticed that high compression ratios are required for practical feasibility of systems. At these ratios, it is evident that forms of information content is lost, such as high-frequency details.

Table 2.1: Bandwidth and storage requirements for 1 hour of SD (720 × 576 pixels) and HD (1920 × 1080 pixels) quality video with 4:2:0 color sampling at different compression ratios.

Resolution	Frames	Raw rate		Target rate		Compression
[pixels]	per sec.	[Mb/s]	[GB/hr]	[Mb/s]	[GB/hr]	factor
1920 × 1080	60	1493	672	20	9.00	75:1
1920 × 1080	30	746	336	10	4.50	75:1
1920 × 1080	15	373	168	5	2.25	75:1
1920 × 1080	10	249	112	3	1.35	83:1
720 × 576	30	149	67.2	4	1.80	37:1
720 × 576	15	74.6	33.6	2	0.90	37:1
720 × 576	10	49.8	22.4	1	0.45	50:1

Discrete Cosine Transform (DCT) coding

Most image and video compression systems in use today utilize the DCT (Discrete Cosine Transform) as the first stage for the compression algorithm, which decorrelates the image coefficients by converting them to a pseudo-frequency domain. Well-known examples of DCT-based compression are the JPEG standard for encoding of still images and the MPEG-2 standard for television broadcasting, which is also used for disc-based storage (DVD). DCT compression systems typically split the full-color image planes in blocks of 8 × 8 pixels, which are individually transformed to the DCT coefficient domain, resembling to some extend the Fourier frequency domain. After transformation, the DCT coefficients are quantized and variable-length coded. For video signals, temporal redundancy is additionally exploited by estimating the movement of objects in the video, using a block-based motion estimator. The current video frame is predicted from the estimated motion, going from one frame to another, while only the difference between this predicted image and the actual image is encoded. This coding principle significantly reduces the required bit rate for a certain visual quality.

The vast majority of current video compression systems employ DCT compression techniques, since it is the best fixed transform for compression. Although this class of systems outperforms all other fixed transforms, DCT coding systems can still show visual artifacts when the bit rate is set too low, leading to blocking, mosquito noise and poor gradient rendering.

Video coding

At present, H.264/MPEG-4 AVC [15] is the most prominent HD video coding standard. It is used for many applications, such as Blu-ray Discs and HD Digital Video Broadcasting (DVB) on terrestrial, cable and satellite (DVB-T, DVB-C and DVB-S, respectively). It is also commonly used for video streaming on the Internet from various well-known websites and on-demand movies[1].

H.264 has evolved from previous DCT-based standards, such as H.263 and MPEG-2. It significantly improves coding performance by adding many new coding features at each processing stage. Similar to previous standards, it uses block-based motion compensation and DCT, but it differs from previous standards in the following ways.

- Multiple previously decoded pictures can be used as a reference, allowing up to 16 reference frames for a single frame, in contrast to the single reference P-pictures, and the double reference B-pictures in MPEG-1/2 and related standards.
- Variable Block-Size Motion Compensation (VBSMC) with up to quarter-pixel precision and advanced multi-tap sub-pixel prediction. Various block sizes can be used, ranging from 16×16, 16×8, 8×16, 8×8, 8×4, 4×8 to 4×4. For the larger block sizes, i.e. 8×8 and above, each block can point to a different reference frame.
- Advanced spatial prediction of blocks from previously decoded neighbouring blocks. Various directions for prediction of DCT coefficients into the new block can be used.
- New integer 4×4 and 8×8 transforms, based on the non-integer 8×8 DCT transform.
- An integrated adaptive deblocking filter within the prediction loop to prevent blocking artifacts.
- A choice of two advanced entropy coders: Context-Adaptive Binary Arithmetic Coding (CABAC) and Context-Adaptive Variable-Length Coding (CAVLC). The CABAC performs better than CAVLC, but is also considerably more complex.

These techniques significantly improve the performance of H.264 over previous standards such as MPEG-2, yielding a factor of two higher compression or even more. It features various techniques to reduce the blocking artifacts associated with DCT-based coding, such as the in-loop deblocking filter, a new transform and flexible block sizes. This performance is obtained at the expense of a 3-4 fold complexity increase. As an alternative to the block-based DCT, wavelet transforms

[1] These services are commercially exploited under the names of YouTube, Vimeo, and iTunes.

are emerging for high-quality video coding such as digital cinema. The next section provides an introduction of the wavelet transforms used in image and video coding.

Discrete Wavelet Transform (DWT) coding

To alleviate the coding artifacts of block-based DCT coding systems, alternative transforms have been proposed. One of the most promising techniques is the wavelet transform, which is based on splitting the video signal into multiple frequency bands. In contrast to the block-based DCT transform, the wavelet transform is applied to the whole image, sequentially addressing both dimensions. This approach splits the image into four frequency bands: LL, LH, HL and HH, with each having half of the horizontal and vertical resolution of the input image. The LL-band is then transformed again, using the same wavelet filters for a predetermined number of levels, creating a hierarchical representation of the coefficients. This particular hierarchical representation using halved frequency bands is called a dyadic wavelet decomposition, which is shown in Figure 2.2 for two hierarchical levels.

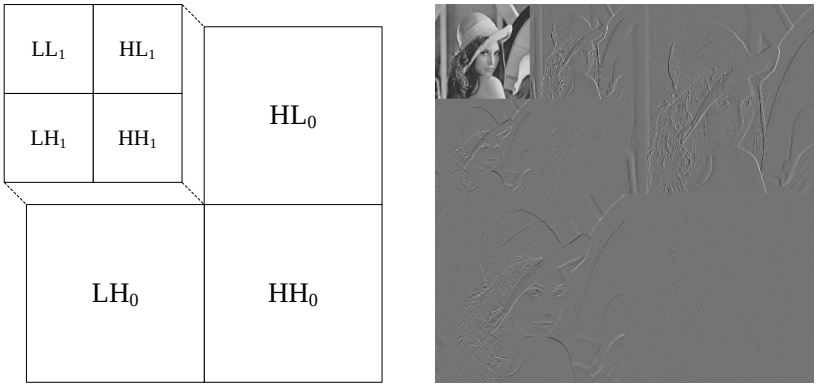


Figure 2.2: Two-level dyadic wavelet decomposition: (a) hierarchical split in wavelet frequency bands and (b) wavelet coefficients for the Lena image. For better display, high-pass wavelet coefficients are extracted at 8-bit resolution and offset with a value of 127 to grey.

From this figure, it can be observed that the wavelet transform effectively decorrelates natural images and compacts most of the image energy in the LL subband. Figure 2.2(b) is also a clear demonstration of the multi-resolution nature of the dyadic wavelet decomposition, thereby providing an inherently natural support

for scalable image and video coding. The following section provides an in-depth introduction to wavelet transforms and their application to scalable coding.

2.3 Wavelets for image and video coding

This section provides an overview of wavelets for image and video coding. It commences with discussing the literature of the wavelet transform, after the lifting framework and integer-to-integer wavelets are presented, which are commonly abbreviated as integer wavelets. Furthermore, also 1D, 2D and multi-level energy correction are explained. Section 2.3 addresses efficient coding techniques for wavelet coefficients.

Brief overview of literature

The multi-resolution representation of images using the Laplacian pyramid [16] is closely related to both subband coding and wavelet decompositions. Mallat [17, 18, 19] used this concept of multi-resolution analysis and wavelets for the application to data compression in image coding, texture discrimination and fractal analysis. He has also introduced the dyadic decomposition structure of Figure 2.2. Daubechies [20] [21] has significantly accelerated coding research, by designing and publishing compact orthogonal wavelet filters, such as the well-known floating-point 9/7 filter [2]. When good orthogonal wavelet transforms became known and available, research emerged to reduce the complexity of these transforms.

One way to reduce the computational complexity of wavelet filters involves replacing the floating-point wavelet filters and arithmetic by custom-designed integer transforms, as proposed by Calderbank *et al.* [22, 23] ((2,2) and (4,4)), Le Gall and Tabatabai [24] and Cohen *et al.* [25] (5/3), Said and Pearlman [26] (S+P) and Antonini *et al.* [27] (9/7). These integer wavelets enable an efficient implementation on fixed-point arithmetic and have the additional benefit of being fully reversible, thereby enabling lossless compression.

Another way to reduce computational complexity of the wavelet transform is achieved by replacing the Finite Impulse Response (FIR) filters by other means of calculation. Sweldens [28, 29] has proposed that wavelet filters can be computed using the lifting framework, where the FIR filter is replaced by a series of so-called predict and update steps. Daubechies and Sweldens [30] later showed that any discrete wavelet transform can be decomposed into a finite sequence of lifting stages.

[2] Due to the publicly available coefficients of the filter design, these filters have become known as Daubechies filters.

The following parts present more details on the specific system aspects: the lifting framework, integer wavelet transforms and multi-level 2D wavelet transforms.

Lifting framework and integer wavelets

Using the lifting framework [28], the wavelet can be implemented with less multipliers and adders than the straightforward FIR implementation. Figure 2.3 shows the principle of the lifting framework. Input samples are split into odd and even samples, after which the even samples are filtered and used to adjust the odd samples in the predict step. Likewise, the odd samples are then filtered and used to adjust the even samples in the update step. Finally, a multiplication is performed on both outputs to normalize signal energy.

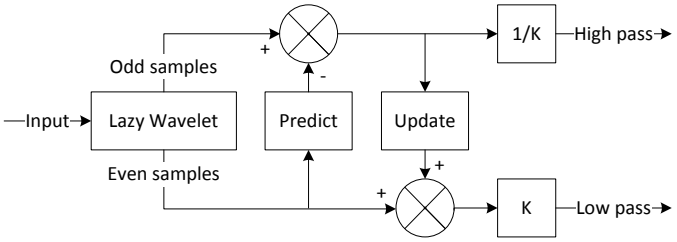


Figure 2.3: Principle of lifting framework for wavelet filtering.

Figure 2.3 shows a single combination of an update and a predict step, which is sufficient to implement the 5/3 wavelet filter. For more complex wavelet filters, more sets of update and predict steps should be cascaded. For example, the 9/7-F integer filter has two sets of update and predict steps. The multiplication factor is needed to normalize signal energy contained in the low- and high-pass bands. The single lifting combination of the 5/3 integer wavelet allows an implementation without any output multiplication. For this filter, the predict and update steps are defined by Equations (2.1) and (2.2), respectively, which are specified by

$$d[n] = d_0[n] - \left\lfloor \frac{s_0[n+1] + s_0[n]}{2} \right\rfloor, \tag{2.1}$$

$$s[n] = s_0[n] + \left\lfloor \frac{d[n] + d[n-1]}{4} + \frac{1}{2} \right\rfloor. \tag{2.2}$$

As can be derived from these formulas, the multiplication factors in the predict and update steps are powers of two, so that the complete 5/3 wavelet can be implemented with only adders and simple bit-shift arithmetic without any mul-

tipliers. This makes the 5/3 integer wavelet a perfect candidate for fixed-point embedded implementations, in which complexity is of primary concern.

The more complex 9/7-F integer wavelet consists of two update and predict steps and is defined by Equations (2.3)–(2.6), such that

$$d_1[n] = d_0[n] + \left\lfloor \frac{203(-s_0[n+1] - s_0[n])}{128} + \frac{1}{2} \right\rfloor, \tag{2.3}$$

$$s_1[n] = s_0[n] + \left\lfloor \frac{217(-d_1[n] - d_1[n-1])}{4096} + \frac{1}{2} \right\rfloor, \tag{2.4}$$

$$d[n] = d_1[n] + \left\lfloor \frac{113(s_1[n+1] + s_1[n])}{128} + \frac{1}{2} \right\rfloor, \tag{2.5}$$

$$s[n] = s_1[n] + \left\lfloor \frac{1817(d_1[n] + d_1[n-1])}{4096} + \frac{1}{2} \right\rfloor. \tag{2.6}$$

Both the 5/3 and 9/7-F integer wavelet are fully reversible and thus lossless. Note that there is no energy correction or scaling factor utilized for these integer wavelets, so that they offer perfect reconstruction.

Multi-level 2D dyadic wavelet decomposition

In wavelet-based image coding, the input image is transformed into the wavelet domain by two-dimensional (2D) separable wavelet filters. The result of the 2D wavelet transform consists of four frequency bands, commonly referred to as the LL, HL, LH and HH bands. The LL band represents the low-pass image in both horizontal and vertical direction and can be seen as a down-scaled version of the original image. The HL, LH and HH bands contain high-pass image information. The LL band still contains a large amount of spatial correlation and therefore, the 2D wavelet transform can be re-applied to this band, even for several times, up to a predetermined typical number of iterations. The chosen number of iterations is a trade-off by remaining resolution of the final LL band, the total number of decomposition stages, the required number of bits to represent the LL coefficients and the coding performance. The dyadic wavelet decomposition for two iterations was already visualized in Figure 2.2.

1D, 2D and multi-level energy correction

Single-level energy correction

To balance the energy between the low- and high-pass output of the 5/3 wavelet, the output scaling factor K present in the lifting framework (see Figure 2.3) is manipulated. For the standard 1D wavelet, $K = \sqrt{2}$ is a good factor for the low-pass

output, which leads to using the reciprocal value for the high-pass output. These factors are well-defined real numbers and thus not suited for fixed-point implementation.

Since the 2D wavelet transform is separable, two 1D wavelet filtering steps are performed in succession. For better precision, we convert both the 1D scaling factors into two alternative 2D scaling factors *after* the 2D transform has completed. Using the initial 1D scaling factor of $K = \sqrt{2}$, this results in the 2D scaling factors, as presented in Table 2.2. These 2D scaling factors can be very efficiently implemented in fixed-point arithmetic with bit-shifting.

Table 2.2: 2D scaling factors for the 5/3 integer wavelet.

		Horizontal	
		Low-pass	High-pass
Vertical	Low-pass	$\sqrt{2} \times \sqrt{2} = 2$	$\frac{1}{\sqrt{2}} \times \sqrt{2} = 1$
	High-pass	$\sqrt{2} \times \frac{1}{\sqrt{2}} = 1$	$\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}} = \frac{1}{2}$

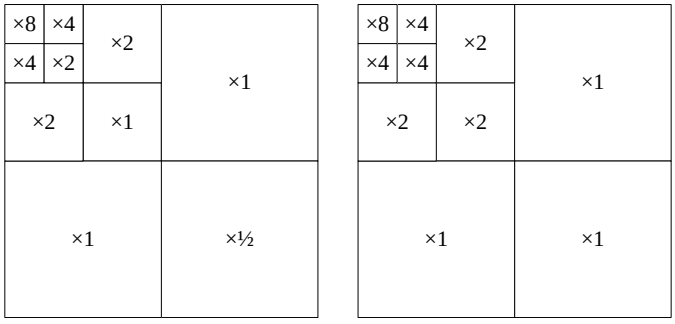
Although scalable wavelet coding is a relatively new paradigm in picture coding applications, the optimization techniques were already applied to early picture coding systems several decades ago, such as the normalization factors used in DCT-based coding.

Despite the simplicity of the HH-band scaling factor of $\frac{1}{2}$, the result of that scaling in fixed-point arithmetic leads to imperfect reconstruction, as the effective result is $y = \lfloor \frac{1}{2}x \rfloor$. For lossy image coding, this is not an issue, but for lossless image coding, a reversible transform is required. As a compromise, the HH-band scaling factor can be omitted, which makes the transform reversible again at the cost of a reduced energy balance, thereby decreasing lossy coding performance. Furthermore, the lossless coding efficiency is most likely also reduced, due to the LL-band scaling factor.

Multi-level 2D energy correction

The dyadic wavelet decomposition consists of multiple filtering operations of the 2D wavelet transform over several scales, as discussed in Section 2.3. It is now also possible to extend the 2D energy corrections to the multi-level framework, which will result in the scaling factors of Figure 2.4(a). These numbers are found to be identical to those proposed by Zhang Li-bao and Wang Ke [31]. The scaling factors

for the energy correction that retains lossless coding, are presented in Figure 2.4(b). This multi-level 2D energy correction is applied after completion of the multi-level wavelet image transformation.



(a) LL×2 and HH×$\frac{1}{2}$ (b) LL×2 only

Figure 2.4: Energy correction for a three-level dyadic wavelet decomposition with (a) both LL- and HH-band scaling and (b) only LL-band scaling.

The energy balance of the 9/7-F integer wavelets is better than the 5/3 integer wavelets, but it is still not fully balanced. We propose to use similar 2D integer scaling factors, as introduced for the 5/3 wavelet, to further improve the energy balance. The resulting scaling factors are found in Table 2.3, which were designed such that the energy balance of the 9/7-F integer wavelet matches the energy balance of the 9/7 floating-point wavelet.

Table 2.3: 2D scaling factors for the 9/7-F integer wavelet.

		Horizontal	
		Low-pass	High-pass
Vertical	Low-pass	$\times\frac{5413}{4096}$	$\times 1$
	High-pass	$\times 1$	$\times\frac{3099}{4096}$

In this section we have considered various aspects of the wavelet transform as used for image and video coding. We have presented the lifting framework and integer-to-integer wavelets, commonly abbreviated as integer wavelets. Finally, we have discussed 1D, 2D and multi-level energy correction strategies to provide

wavelet coefficients with good energy balance. In the following section, we discuss the next step in the scalable video coding structure, addressing efficient coding techniques for wavelet coefficients.

Wavelet coefficient coding

After wavelet transformation discussed in the previous section, let us now turn to the following stage, concentrating on the efficient coding of the wavelet coefficients. Over the years, several wavelet coefficient encoding algorithms have been proposed in literature. Most of these algorithms utilize the multi-resolution representation of images and the property that the high-pass wavelet coefficients are sparsely distributed non-zero coefficient values, which also have inter-band correlations. Since wavelets maintain a certain form of spatial information in their frequency analysis, unlike the Fast Fourier Transform (FFT), the correlation between bands of the same resolution and/or correlation across resolutions, can be exploited for higher coding efficiency. An example of such correlation phenomena is created by a sharp signal transient that introduces significant wavelet coefficients across different frequency scales at approximately the same spatial location.

Lewis and Knowles [32] were the first to exploit the inter-resolution correlations of the multi-resolution representation of images. The EZW (Embedded Zerotree Wavelet) algorithm proposed by Shapiro [33, 34, 35] exploits these inter-resolution correlations, by introducing the concept of zerotrees. Zerotrees can be used to effectively encode large regions of insignificant wavelet coefficients across spatial resolutions. The coding algorithm provides progressive transmission, encoding the largest coefficients first, which are organized and then processed from the highest bit plane downwards [36]. More specifically, the coding commences with the most significant bit plane and codes the coefficients present in that bit plane, as a sequence of zeros and ones, followed by a bit plane lower, and so on. Consequently, each following bit-plane layer refines the accuracy of the description of the coefficients, which refers to progressive transmission of quality. For compression, these refinement layers allow the bitstream to be terminated at any given bit, while still providing a suitable partial image reconstruction. This concept has the attractive feature that it yields the best possible image reconstruction for that bit rate, related to the point of terminating the bitstream. This attractive truncation feature can be exploited at any stage in the video communication chain, e.g. in the encoder, decoder and in post-processing. Furthermore, every additional bit encoded/decoded from the bitstream layer increases the quality of the image. Shapiro improved EZW by proposing a fast technique for identifying zerotrees [37]. Creusere added regional decompression at a minor loss in compres-

sion efficiency [38], also improving robustness in case of transmission errors [39].

Said and Pearlman improved on the concept of zerotrees [40], by introducing a set partitioning algorithm, which has later evolved to the well-known SPIHT [41] (Set Partitioning in Hierarchical Trees) algorithm proposed by the same authors. SPIHT outperforms EZW in terms of coding efficiency and complexity and is often used as a quality reference in wavelet image compression literature.

Outside the wavelet domain, quadtree partitioning has been used for image coding with variable results results [42, 43]. However, for wavelet coding, Islam and Pearlman [44] and Islam *et al.*[45] applied quadtree partitioning to the wavelet coefficients in their proposed SPECK algorithm. SPECK (Set Partition Embedded bloCK) exploits the sparse distribution of significant wavelet coefficients at various bit planes like EZW and SPIHT, but isolates the significant coefficients by performing quadtree partitioning individually for each of the wavelet bands (blocks). At a first glance, it seems unfortunate that the correlation of coefficients between frequency and/or resolution bands is not utilized. However, the processing of individual bands has the benefit of being fully based on local data only, allowing highly efficient cache memories usage, which speeds up the algorithm significantly. Moreover, it offers the possibility of parallel implementations, in which different frequency bands and resolutions can be processed simultaneously. Despite missing the cross-frequency band correlations and cross-resolution correlations, SPECK still rivals EZW and SPIHT in coding performance. Wheeler and Pearlman [46] has combined concepts from SPIHT and SPECK, in order to slightly improve coding performance, at the cost of additional computational complexity.

Finally, EBCOT (Embedded Block Coding with Optimal Truncation) was proposed by Taubman [47, 48] and later adopted in the JPEG 2000 standard [49, 50]. EBCOT provides excellent compression performance and includes resolution scalability, SNR scalability and a random-access capability. However, to provide all of these features simultaneously, the algorithmic complexity of EBCOT is significant, and processing single bit planes requires several coding passes. The random-access capability, as required by the JPEG 2000 committee, was also added to SPIHT by Wheeler and Pearlman [51].

Fortunately, JPEG 2000 has been adopted for digital cinema, which has revived the importance of the standard and recently another high-end application of JPEG 2000 has emerged in the area of high-quality (Ultra-HD) professional video surveillance [52, 53]. Since the application area of this thesis is more related to high-volume surveillance at broadcast resolution, we will adopt the low-complexity SPIHT and SPECK codecs as a basis for our exploration. The following sections will discuss these two codecs in more detail, and provide an elaborate example to illustrate the operation of both algorithms.

2.4 SPIHT coding (Set Partitioning in Hierarchical Trees)

This section gives an elaborate overview of the SPIHT coding technique proposed by Said and Pearlman [41], together with an example to understand its operation. SPIHT coefficient coding is based on three concepts. First, the wavelet coefficients are ordered by magnitude, following a spatial ordering algorithm that is duplicated at the decoder. Second, refinement bits are transmitted in ranking order of the bit planes, from most significant to least significant bits. Third, the similarity of significant wavelet coefficients is exploited, at different locations, both locally and across different scales. These three concepts will be explored in more detail in the following paragraphs.

Transmission of coefficients

Said and Pearlman show that for a unitary transform, coefficients with a larger magnitude contribute most to decreasing the Mean Square Error (MSE) of the decoded image and that these coefficients should be transmitted first because of their larger information content. This concept follows the progressive transmission method proposed by DeVore *et al.* [54].

In coding, coefficients are commonly represented in a fixed-point binary format. In this case, magnitude ordering of coefficients can be obtained by ordering them, based on the number of bits required to represent the absolute value of the coefficient, transmitting bits from the most significant to the least significant bit plane. Figure 2.5 shows the binary representation of 16 coefficients, ordered by magnitude. It should be noted that these coefficients can have varying locations and signs. They are solely ordered based on the absolute value of the coefficient. We adopt the definition that the bits in the highest and the lowest bit planes are considered to be the most and least significant, respectively.

For successful transmission and decoding of these coefficients, several pieces of information need to be transmitted: (1) the highest utilized bit-plane index of the most significant coefficient, (2) information regarding the ordering of the coefficients, (3) the number of coefficients that are significant when observed at a certain bit plane, and (4) sign information and refinement bits.

All bits in a row have the same contribution in reducing the MSE, and therefore the bits can be transferred sequentially per bit plane as indicated by the arrows in Figure 2.5. Furthermore, the '0's and the first '1' of each coefficient are not transmitted, as they can be inferred from the transmission order and number of significant coefficients per bit plane.

An algorithm to encode the coefficients based on the magnitude ordering is

order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
position	(0,0)	(1,1)	(1,0)	(0,1)	(1,2)	(2,3)	(0,2)	(0,3)	(2,1)	(3,2)	(1,3)	(2,0)	(3,0)	(3,3)	(3,1)	(2,2)
sign	+	+	+	+	+	−	+	+	−	+	+	−	−	+	−	+
MSB: bit 5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bit 4	→		1	1	0	0	0	0	0	0	0	0	0	0	0	0
bit 3	→	→	→		1	1	1	1	0	0	0	0	0	0	0	0
bit 2	→	→	→	→	→	→	→		1	1	1	1	1	1	1	1
bit 1	→	→	→	→	→	→	→	→	→	→	→	→	→	→	→	
LSB: bit 0	→	→	→	→	→	→	→	→	→	→	→	→	→	→	→	

Figure 2.5: Binary representation of the magnitude-ordered coefficients.

given in Algorithm 2.1. With this algorithm, images of good quality can be reconstructed, even when a small fraction of the coefficients have been transmitted. However, the transmission of locations in the magnitude sorting pass is very inefficient and therefore SPIHT proposes a sophisticated approach to implicitly convey the sorting information. This is discussed next.

Algorithm 2.1 Progressive transmission of magnitude-ordered coefficients

1: Transmit $maxBitPlane = \lfloor log_2(max(abs(coefficients)) \rfloor$
2: **for** $bitPlane$ = $maxBitPlane$ to 0, step -1 **do**
   ↻ *Sorting pass:*
3: Determine new significant coefficients for this $bitPlane$
4: Transmit number of new significant coefficients
5: Transmit location of new significant coefficients
6: Transmit sign of new significant coefficients
   ↻ *Refinement pass:*
7: Transmit bits at current $bitPlane$ of previously significant coefficients
8: **end for**

Set partitioning sorting algorithm

SPIHT does not transmit the order of wavelet coefficients explicitly, but embeds a magnitude-based ordering strategy within the algorithm by implementing a coefficient sorting algorithm. If both the encoder and decoder utilize the same sorting

algorithm, the decoder can follow the coefficient ordering by observing the results of the sorting decisions of the encoder.

The sorting decisions in the encoder are based on a magnitude comparison function which determines if a certain coefficient is significant or not. The coefficient c at location (i, j) is considered significant for a certain bit plane n, if $|c_{i,j}| \geq 2^n$. The set significance S for a certain group of coefficients τ at bit plane n is consequently determined by:

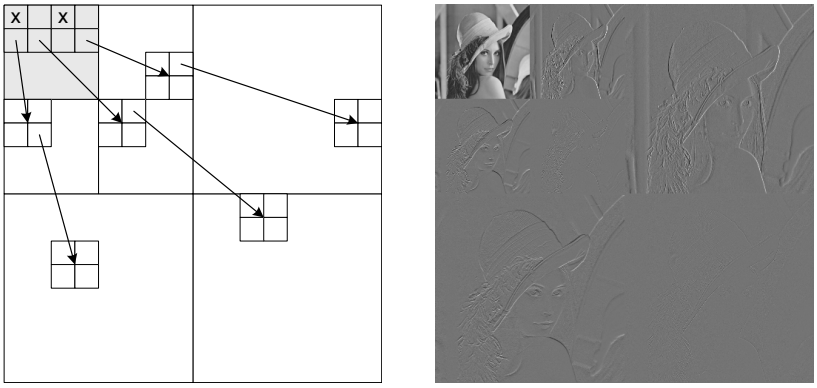
$$S_n(\tau) = \begin{cases} 1, & \text{if } \max\limits_{(i,j)\in\tau} \{|c_{i,j}|\} \geq 2^n, \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

If a certain subset τ is considered insignificant, $S_n(\tau) = 0$, which means that all coefficients in this subset are insignificant. If the subset is considered significant, $S_n(\tau) = 1$, which indicates that at least one or more coefficients in this subset are significant. To determine which coefficient magnitudes exceed bit plane n, both the encoder and decoder split the subset into new subsets following a fixed strategy, while each of the subsets is tested in the same way, until each significant coefficient is identified. This partitioning into subsets is based on spatial orientation trees.

Spatial orientation trees

The spatial orientation tree, as displayed in Figure 2.6(a), is used to perform subset partitioning, and describes the relation between wavelet coefficients. Through this tree, also location information of sets and individual coefficients are implicitly transferred from the encoder to the decoder. Its hierarchical representation is based on the fact that coefficients at approximately the same spatial location have a strong self-similarity (correlation) between subbands. This can be easily understood if we look at the low-activity areas in Figure 2.6(b), where large areas of near-zero coefficients are replicated at lower levels of the pyramidal structure at approximately the same spatial location. Similar reasoning holds for areas with high activity, such as strong edges and boundaries.

The tree is defined in such a way that each node either has no offspring (the leafs at the lowest level of the pyramid), or an offspring of 4 children forming a group of 2×2 adjacent coefficients. The arrows in Figure 2.6(a) indicate this relationship. At the highest pyramid level, indicated by grey shading in the figure, coefficients are the tree roots. Their offspring rule is different and the top-left node in a group of 4 does not have any offspring, which is indicated in the figure by the '$\times$' symbol. Within the spatial orientation tree, several sets of coordinates are defined, which are listed in Table 2.4.



(a) (b)

Figure 2.6: (a) Examples of parent-offspring relations in the spatial orientation tree. The grey area indicates the highest pyramid level, where the elements marked with an '$\times$' symbol have no offspring. (b) Wavelet coefficients describing an area of self-similarity (e.g. the hat) can be found within different subbands, visual as remaining edges.

Table 2.4: Set of coordinates defined in SPIHT.

Set	Contains
$\mathcal{H}$	Set of coordinates of all spatial orientation tree roots
$\mathcal{O}(i,j)$	Set of coordinates of all offspring of node (i,j)
$\mathcal{D}(i,j)$	Set of coordinates of all descendants of node (i,j)
$\mathcal{L}(i,j)$	$\mathcal{D}(i,j) - \mathcal{O}(i,j)$ (All descendants that are not offspring)

The offspring of all nodes, except for the highest and lowest pyramid levels, is defined for node (i,j) by:

$$\mathcal{O}(i,j) = \{(2i,2j), (2i+1,2j), (2i,2j+1), (2i+1,2j+1)\}. \tag{2.8}$$

Now that relations are established by the spatial orientation tree and we have a notation for coordinate sets in SPIHT, the following set partitioning rules can be defined:

1. The initial partition is formed with the sets (i,j) and $\mathcal{D}(i,j)$, for all $(i,j) \in \mathcal{H}$,
2. If $\mathcal{D}(i,j)$ is significant, then it is partitioned into $\mathcal{L}(i,j)$ and 4 coefficient locations (k,j), with $(k,l) \in \mathcal{O}(i,j)$,
3. If $\mathcal{L}(i,j)$ is significant, it is partitioned into the four sets $\mathcal{D}(k,l)$, with $(k,l) \in \mathcal{O}(i,j)$.

Previous paragraphs have defined the set partitioning algorithm, spatial orientation trees and the set partitioning rules. They all come together in the SPIHT coding algorithm, which is discussed in the following section.

SPIHT coding algorithm

To synchronize encoder and decoder, both need to keep track of the processing order of the subsets. In SPIHT, the significance information is stored in three ordered lists: a List of Insignificant Pixels (LIP), a List of Significant Pixels (LSP) and a List of Insignificant Sets (LIS), in which entries are stored by their location (i, j). The LIP and LSP memorize individual coefficient positions, while the LIS stores sets, either $\mathcal{D}(i, j)$ or $\mathcal{L}(i, j)$, which are differentiated in the LIS as sets of type A and B, respectively. This is denoted by $(i, j)\mathbf{t}$, with $\mathbf{t}$ the type of the set.

The coding algorithm of SPIHT is similar to Algorithm 2.1, which processes coefficients per bit plane, using a sorting and a refinement pass. In contrast to Algorithm 2.1, locations of significant coefficients are not explicitly transmitted, and SPIHT utilizes the set partitioning sorting algorithm and spatial orientation trees to synchronize the encoder and the decoder.

This synchronization is performed by redefining the sorting and refinement passes. In the sorting pass, the significance of all coefficients in the LIP are tested, and newly significant coefficients are moved to the LSP. The same is performed for the LIS, and all insignificant sets are re-evaluated, and if found significant, they are removed from the LIS and partitioned into subsets. Insignificant subsets remain in the LIS, and once the leafs are reached, the individual coefficients are either added to the LIS or the LIP, depending on their significance. In the refinement pass, the existing coefficients in the LSP are processed, and their bits are send out in the pre-defined order of the spatial orientation trees.

The complete coding algorithm of SPIHT is represented in pseudo-code in Algorithm 2.2, with two parts of the sorting pass detailed in Algorithms 2.3 and 2.4. It should be noted that during the sorting pass, elements are added to the LIS and these new elements should be processed during the same sorting pass.

The decoding algorithm is nearly identical to the encoding algorithm, but with send actions replaced by receive actions. In this way, the decoder follows an identical execution path as the encoder and has the same data in its LIP, LSP and LIS lists. Additionally, the decoder performs a refinement step during the creation of newly significant coefficients. It is known that when a coefficient is moved to the LSP (newly significant), its value $|c_{i,j}|$ is between 2^n and 2^{n+1}, so that the decoder stores the value 1.5×2^n. During refinement of these significant coefficients, this behavior is repeated for lower bit planes.

Algorithm 2.2 SPIHT coding algorithm using the set partitioning sorting algorithm and spatial orientation trees.

↷ *Initialization*
Transmit $maxBitPlane = \lfloor log_2(max(abs(coefficients))) \rfloor$
Clear the LSP, add $(i,j) \in \mathcal{H}$ to the LIP ▷ Set partitioning rule 1
Add $(i,j) \in \mathcal{H}$ with descendants to the LIS ▷ Set partitioning rule 1
for n = *maxBitPlane* to 0, step -1 **do**
  ↻ *Sorting pass:*
  Perform sorting pass on elements in the LIP ▷ See Algorithm 2.3
  Perform sorting pass on elements in the LIS ▷ See Algorithm 2.4
  ↻ *Refinement pass:*
  for each entry of the LSP not added in the last sorting pass **do**
    Transmit n-th bit of $|c_{i,j}|$ ▷ Refine already significant coefficients
  end for
end for

Algorithm 2.3 SPIHT sorting of elements in LIP.

for all entries in the LIP **do**
  Transmit $S_n(i,j)$ ▷ So that the decoder can follow
  if $S_n(i,j) = 1$ **then** ▷ If coefficient is newly significant then
    Transmit sign of $c_{i,j}$
    Move coefficient location to LSP
  end if
end for

SPHIT encoding example

To clarify the working of SPIHT and its lists, a small example is presented using a set of 4×4 coefficients. Table 2.5(a) shows the arrangement of coefficients, and Table 2.5(b) the contents of the lists used by SPIHT at the start of the algorithm.

↷ *Initialization*
The number of bit planes is $n = \lfloor log_2(29) \rfloor = 4$, which is transmitted to the decoder, and the lists are initialized to the state shown in Table 2.5(b).

Algorithm 2.4 SPIHT sorting of elements in LIS.

for all entries in the LIS **do**
  if entry is type A **then** ▷ Type A sets represent $\mathcal{D}(i,j)$
    Transmit $S_n(\mathcal{D}(i,j))$ ▷ So that the decoder can follow
    if $S_n(\mathcal{D}(i,j)) = 1$ **then** ▷ If set is newly significant then
      for each $(k,l) \in \mathcal{O}(i,j)$ **do** ▷ Set partitioning rule 2
        Transmit $S_n(k,l)$ ▷ So that the decoder can follow
        if $S_n(k,j) = 1$ **then** ▷ If coefficient is newly significant then
          Transmit sign of $c_{i,j}$
          Add coefficient location (k,j) to LSP
        else
          Add coefficient location (k,j) to LIP
        end if
      end for
      if $\mathcal{L}(i,j) \neq \emptyset$ **then** ▷ Set partitioning rule 2
        Move (i,j) to end of LIS as type B
      else
        Remove (i,j) from the LIS
      end if
    end if
  end if
  if entry is type B **then** ▷ Type B sets represent $\mathcal{L}(i,j)$
    Transmit $S_n(\mathcal{L}(i,j))$ ▷ So that the decoder can follow
    if $S_n(\mathcal{L}(i,j)) = 1$ **then** ▷ If set is newly significant then
      Add each $(k,l) \in \mathcal{O}(i,j)$ to the LIS as type A ▷ Set partitioning rule 3
      Remove (i,j) from the LIS
    end if
  end if
end for

	0	1	2	3
0	29	15	6	-5
1	12	10	-4	9
2	-5	2	1	-1
3	-6	3	-1	0

(a)

List	Contents
LIP	{ (0,0), (0,1), (1,0), (1,1) }
LIS	{ (0,1)**A**, (1,0)**A**, (1,1)**A** }
LSP	{ empty }

(b)

Table 2.5: Example of the SPIHT encoding process: (a) input coefficients, (b) SPIHT lists and their contents at the start of the algorithm.

↻ *First sorting pass*
First the LIP is processed, and $c_{0,0}$ is found to be significant. Bit string '10' is transmitted to indicate its significance and its positive sign and $(0,0)$ is moved to the LSP. The other three coefficients in the LIP are insignificant, for which '000' is transmitted. Next the LIS is processed and all three sets are found to be insignificant, for which bit string '000' is transmitted.

→ *Status*
So far '10000000' was transmitted, so that the contents of the lists is as follows.

LIP	{ $(0,1)$, $(1,0)$, $(1,1)$ }
LIS	{ $(0,1)\mathbf{A}$, $(1,0)\mathbf{A}$, $(1,1)\mathbf{A}$ }
LSP	{ $(0,0)$ }

↻ *First refinement pass*
No refinement pass is performed during the first pass, as the LSP is empty at the start.

↻ *Second sorting pass*
Now $n = 3$ and the significance threshold $T = 2^3 = 8$. First the LIP is processed, and all elements are significant. Bit string '10' is transmitted for each of them (significant=1, sign=0) and they are moved to the LSP. Next the LIS is processed and set $(0,1)\mathbf{A}$ is significant because one of its values is significant ($c_{1,3} = 9$), which is transmitted by a '1'. The offspring of set $(0,1)$ is processed and the first three values ($c_{0,2}$, $c_{0,3}$ and $c_{1,2}$) are insignificant and added to the LIP and '000' is transmitted. Coefficient $c_{1,3}$ is significant and positive, thus '10' is transmitted. Its location $(1,3)$ is added to the LSP. The remaining sets $(1,0)\mathbf{A}$ and $(1,1)\mathbf{A}$ are insignificant and a '0' is transmitted for each. Since $\mathcal{L}(0,1) = \emptyset$, set $(0,1)\mathbf{A}$ is removed from the LIS.

↻ *Second refinement pass*
The only location that was already in the LSP at the start of the second pass is $(0,0)$. The fourth bit ($n = 3$) from the Least Significant Bit (LSB) of $c_{0,0}$ is transmitted and since $|c_{0,0}| = 29 =$`0b11101`, the underlined bit of this string is transmitted, which is a '1'.

→ *Status*
During the second pass '101010100010001' was transmitted, the lists are below.

LIP	{ (0,2), (0,3), (1,2) }
LIS	{ (1,0)**A**, (1,1)**A** }
LSP	{ (0,0), (0,1), (1,0), (1,1), (1,3) }

↻ *Third sorting pass*
Now $n = 2$ and $T = 4$. For the LIP, move $(0,2)$, $(0,3)$, $(1,2)$ → LSP and transmit '10', '11' and '11'. For the LIS, set $(1,0)\mathbf{A}$ is significant, transmit '1' and process its offspring: $(2,0)$, $(2,1)$, $(3,0)$ and $(3,1)$. Coefficient $(2,0)$ is significant → LSP, transmit '11', $(2,1)$ is insignificant → LIP, transmit '0', $(3,0)$ is significant → LSP, transmit '11', $(3,1)$ is insignificant → LIP, transmit '0'. Set $\mathcal{L}(1,0) = \emptyset$, so that $(1,0)\mathbf{A}$ is removed from the LIS. Set $(1,1)\mathbf{A}$ is insignificant, transmit '0'.

↻ *Third refinement pass*
Transmit the 3rd bit ($n = 2$) from the LSB of locations $(0,0)$ (29 = 0b11101), $(0,1)$ (15 = 0b01111), $(1,0)$ (12 = 0b01100), $(1,1)$ (10 = 0b01010), $(1,3)$ (9 = 0b01001), which cascades to '11100'.

→ *Status*
During the third pass '1011111110110011100' was transmitted, leading to the following lists.

LIP	{ (2,1), (3,1) }
LIS	{ (1,1)**A** }
LSP	{ (0,0), (0,1), (1,0), (1,1), (1,3), (0,2), (0,3), (1,2), (2,0), (3,0) }

This process continues until $n = 0$ and all coefficients are processed. Using the created bitstream, we will give an example of the decoding process in SPIHT.

SPIHT decoding example

As mentioned before, the decoding process follows the exact execution path of the encoder, which has the same data in its LIP, LSP and LIS lists, so that this information will not be repeated. The decoder does perform an additional step during the creation of newly significant coefficients by estimating the expected value between bit-plane boundaries. To visualize the decoding of the bitstream by the decoder, the contents of the output are given in Table 2.6 for the same moments

in time as the encoder example above. Table 2.7 gives the results of the SPIHT encoding/decoding process after decoding 42 bits. It can be seen that the average absolute error is only unity.

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(a)

24	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(b)

24	12	0	0
12	12	0	12
0	0	0	0
0	0	0	0

(c)

28	12	0	0
12	12	0	12
0	0	0	0
0	0	0	0

(d)

28	12	6	-6
12	12	-6	12
-6	0	0	0
-6	0	0	0

(e)

30	14	6	-6
14	10	-6	10
-6	0	0	0
-6	0	0	0

(f)

Table 2.6: SPIHT decoded output after: (a) initialization, (b) 1st sorting pass, (c) 2nd sorting pass, (d) 2nd refinement pass, (e) 3rd sorting pass, (f) 3rd refinement pass.

29	15	6	-5
12	10	-4	9
-5	2	1	-1
-6	3	-1	0

(a)

30	14	6	-6
14	10	-6	10
-6	0	0	0
-6	0	0	0

(b)

1	-1	0	-1
2	0	-2	1
-1	2	-1	1
0	3	1	0

(c)

Table 2.7: Results of the SPIHT encoding/decoding process: (a) input, (b) output after decoding 42 bits, (c) difference between input and output.

2.5 SPECK codec (Set Partition Embedded bloCK)

In line with the previous section, this section will give a brief overview of the SPECK codec as proposed by Islam *et al.* [44, 45], together with a small example. SPECK utilizes the sparse distribution of significant wavelet coefficients at various bit planes just like SPIHT, but isolates the significant coefficients by performing

quadtree partitioning on each of the wavelet bands (blocks). The following paragraphs will discuss the similarities and the differences between SPIHT and SPECK, following the same structure used to describe SPIHT in Section 2.4.

Transmission of coefficients

Similar to SPIHT, coefficients with larger magnitudes are transmitted first, as they contribute most to decreasing the MSE of the decoded image. Transmission follows a similar magnitude ordering scheme as SPIHT and the same information has to be transmitted: (1) the highest utilized bit-plane index of the most significant coefficient, (2) information regarding the ordering of the coefficients, (3) the number of coefficients that are significant when observed at a certain bit plane, and (4) sign information and refinement bits.

Set partitioning sorting algorithm

The sorting decisions in the SPECK encoder are based on the same magnitude comparison function as in SPIHT, which determines if a certain coefficient or set of coefficients is significant or not. For clarity, we recall Equation (2.7) of SPIHT below, which defines that the set significance S for a certain group of coefficients τ at bit plane n is determined by:

$$S_n(\tau) = \begin{cases} 1, & \text{if } \max\limits_{(i,j)\in\tau} \{|c_{i,j}|\} \geq 2^n, \\ 0, & \text{otherwise,} \end{cases}$$

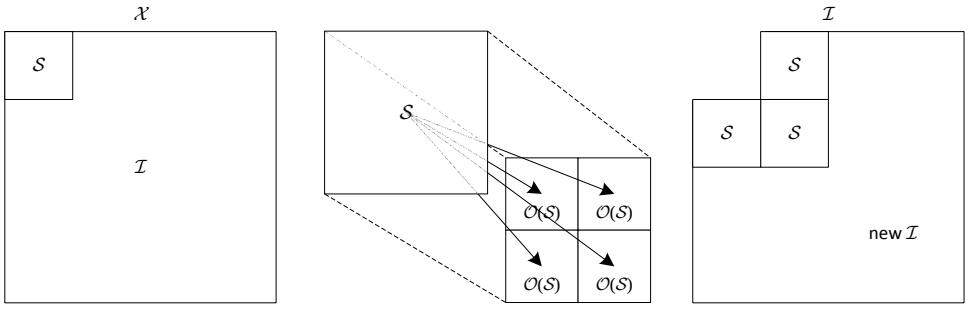
where $c_{i,j}$ denotes the value of the coefficient at location (i, j).

Up this point, SPIHT and SPECK are very similar, utilizing the same concept of magnitude-ordered coefficient transmission and employing an identical significance test for coefficients and sets. However, SPECK deviates from SPIHT in the way how the sets are defined. Sets in SPECK are based on rectangular coefficient regions, describing corresponding regions of the image, which are referred to as sets of type $\mathcal{S}$. A coefficient set $\mathcal{S}$ can be of varying dimensions, based on the size of the original image, and the subband level of the pyramid. The size of coefficient set $\mathcal{S}$ is defined as the cardinality $\mathcal{C}$, or in other words, the number of elements in the set. This is expressed as:

$$\text{size}(\mathcal{S}) = \mathcal{C}(\mathcal{S}) \equiv |\mathcal{S}|. \tag{2.9}$$

During processing, sets of different sizes are formed, including sets of only a single coefficient, where $\text{size}(\mathcal{S}) = 1$. Besides sets of type $\mathcal{S}$, another coefficient set

exists of type $\mathcal{I}$. This set comprises of the coefficients of the image, with a square region removed from the top-left corner. An example of an image $\mathcal{X}$ divided into the two typical sets $\mathcal{S}$ and $\mathcal{I}$, is shown in Figure 2.7(a), forming the initial partitioning of the image[3]. The sets of coordinates defined in this initial partitioning are given in Table 2.8.



(a) Initial partitioning (b) Quadtree partitioning (c) Octave band partitioning

Figure 2.7: SPECK Partitioning: (a) Initial partitioning of image $\mathcal{X}$ into sets $\mathcal{S}$ and $\mathcal{I}$, (b) quadtree partitioning of set $\mathcal{S}$, creating four offspring sets $\mathcal{O}(\mathcal{S})$ and (c) octave band partitioning of set $\mathcal{I}$, creating three sets $\mathcal{S}$ and a new $\mathcal{I}$ set.

Table 2.8: Sets of coordinates defined in SPECK.

Set	Contains
$\mathcal{X}$	Complete image
$\mathcal{S}$	Root set containing top-left region at the highest pyramid level of the dyadic decomposition
$\mathcal{I}$	Remainder of image: $\mathcal{I} = \mathcal{X} - \mathcal{S}$

Another difference between SPIHT and SPECK is the way how these sets are partitioned in the case of significance. This partitioning into subsets is based on quadtree partitioning for sets of type $\mathcal{S}$ and octave band partitioning for sets of type $\mathcal{I}$.

[3]It should be noted that the flow of partitioning in SPECK follows the inverse order of that dyadic wavelet decomposition of images, where the image is initially split in four quadrants, and then the upper-left is again split, and so on.

Quadtree partitioning

If a set $\mathcal{S}$ is found to be significant at bit plane n, it is partitioned into an offspring of four children. The division is based on quadtree partitioning, resulting in four subsets $\mathcal{O}(\mathcal{S})$ of approximately one-fourth the size of set $\mathcal{S}$. This quadtree partitioning is visualized in Figure 2.7(b).

Each of the subsets $\mathcal{O}(\mathcal{S})$ is considered as a new set $\mathcal{S}$, and processed recursively, until individual coefficients are reached at the leafs of the quadtree, where size$(\mathcal{S}) = 1$. The quadtree partitioning is motivated by the fact that we want to specify the location of significant coefficients fastly and efficiently, focusing quickly on areas of high-energy content. Once all significant coefficients are found and no other sets are considered significant at the current bit plane n, the set $\mathcal{I}$ is partitioned through octave band partitioning.

Octave band partitioning

After completing sets of type $\mathcal{S}$, set $\mathcal{I}$ is tested against the same significance magnitude. If it is found to be significant, it is partitioned using octave band partitioning, which is illustrated in Figure 2.7(c). Set $\mathcal{I}$ is hereby split into four sets: three of type $\mathcal{S}$ and a new set $\mathcal{I}$. The size of each of the three sets $\mathcal{S}$ is identical to the area of the already processed section of image $\mathcal{X}$. The remaining coordinates are placed in the new $\mathcal{I}$ set. Once the lowest level of the pyramidal structure is reached, set $\mathcal{I}$ has become an empty set.

The reasoning behind this partitioning scheme is to follow the pyramidal structure of the dyadic wavelet transform and to exploit the fact that most of the energy is concentrated at the highest levels (i.e. at the top-left LPF corner) of the transform.

The definition of the partitioning rules and the ordering algorithm, enables us to fuse them in the SPECK coding algorithm, which is discussed in the following section.

SPECK coding algorithm

The complete coding algorithm of SPECK is represented in pseudo-code in Algorithms 2.5- 2.9 and visualized in Figure 2.8. In the figure, the grey blocks represent the individual algorithms and the big white arrows represent one algorithm calling another.

SPECK utilizes two lists to keep track of the processing order of the subsets, which in turn synchronize the encoder and decoder. The significance information is stored in two ordered lists: a List of Significant Pixels (LSP) and a List of

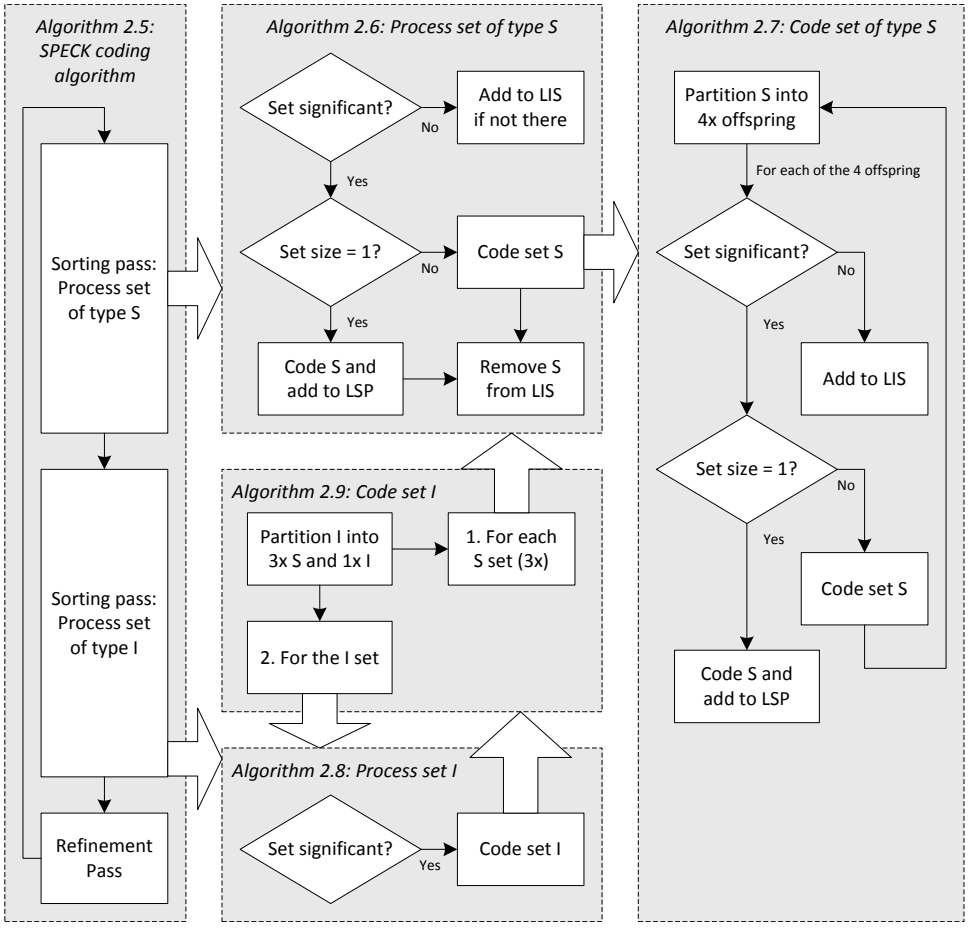


Figure 2.8: Graphical representation of the coding algorithms in SPECK. The large white arrows indicate one algorithm (recursively) calling another.

Insignificant Sets (LIS). The LSP stores individual coefficient positions of already significant coefficients, while the LIS stores sets of type $\mathcal{S}$ of varying sizes which are not yet considered significant for n.

The coding algorithm is divided into a sorting and a refinement pass per bit plane n. Both the encoder and the decoder follow the same sorting and set partitioning rules, thereby synchronizing the encoder and the decoder and implicitly conveying coefficient position information of significant positions and insignificant sets.

The main algorithm (Algorithm 2.5) contains the functionality that calls the sorting and refinement passes for each of the bit planes. Sorting for sets of type $\mathcal{S}$ is performed by the ProcessS and CodeS algorithms, represented by Algorithm 2.6 and Algorithm 2.7, respectively. After the sets of type $\mathcal{S}$ are processed, the $\mathcal{I}$ set is processed and coded using the ProcessI and CodeI algorithms, defined by Algorithm 2.8 and Algorithm 2.9, respectively.

Algorithm 2.5 SPECK coding algorithm

↷ *Initialization*
Partition image $\mathcal{X}$ into root set $\mathcal{S}$ and set $\mathcal{I} = \mathcal{X} - \mathcal{S}$ ▷ See Figure 2.7a
Transmit $maxBitPlane = \lfloor log_2(max(abs(coefficients))) \rfloor$
Add set $\mathcal{S}$ to the LIS
Set LSP $= \emptyset$
for n = $maxBitPlane$ to 0, step -1 **do**
  ↻ *Sorting pass:*
  for each set $\mathcal{S} \in$ LIS in increasing order of size($\mathcal{S}$) **do**
    Process set $\mathcal{S}$ ▷ See Algorithm 2.6
  end for
  Process set $\mathcal{I}$ ▷ See Algorithm 2.8
  ↻ *Refinement pass:*
  for each entry of the LSP not added in the last sorting pass **do**
    Transmit n-th bit of $|c_{i,j}|$ ▷ Refine already significant coefficients
  end for
end for

Algorithm 2.6 SPECK process set of type $\mathcal{S}$

Transmit $S_n(\mathcal{S})$ ▷ So that the decoder can follow
if $S_n(\mathcal{S}) = 1$ **then** ▷ If set is newly significant then
  if size$(\mathcal{S}) = 1$ **then** ▷ Is set one coefficient?
    Transmit sign of $\mathcal{S}$ and add $\mathcal{S}$ to LSP
  else
    Code set $\mathcal{S}$ ▷ Includes partitioning of set $\mathcal{S}$, see Algorithm 2.7
  end if
  if $\mathcal{S} \in$ LIS **then**
    Remove $\mathcal{S}$ from LIS ▷ Added to LSP or partitioned
  end if
else
  if $\mathcal{S} \notin$ LIS **then**
    Add $\mathcal{S}$ to LIS ▷ Set is not significant, add to LIS if not there
  end if
end if

Algorithm 2.7 SPECK code set of type $\mathcal{S}$

```
Partition S into four subsets O(S)                          ▷ See Figure 2.7b
for each set O(S) do
    Transmit S_n(O(S))                                      ▷ So that the decoder can follow
    if S_n(O(S)) = 1 then                                   ▷ If subset is newly significant then
        if size(O(S)) = 1 then                              ▷ Is set one coefficient?
            Transmit sign of O(S) and add O(S) to LSP
        else
            Code set O(S)                                   ▷ Recursive call of this algorithm
        end if
    else
        Add O(S) to LIS                                     ▷ Subset is not significant, add to LIS
    end if
end for
```

Algorithm 2.8 SPECK process set of type $\mathcal{I}$

```
Transmit S_n(I)                                             ▷ So that the decoder can follow
if S_n(I) = 1 then                                          ▷ If set is newly significant then
    Code set I                                              ▷ Includes partitioning of set I, see Algorithm 2.9
end if
```

Algorithm 2.9 SPECK code set of type $\mathcal{I}$

```
Partition I into four subsets, three S and one new I        ▷ See Figure 2.7c
for each of the three sets S do
    Process set S                                           ▷ Recursive call to Algorithm 2.6
end for
Process set I                                               ▷ Recursive call to Algorithm 2.8
```

As can be seen in the figure and the pseudo-code algorithms, coding is recursive for several functions. For example, the CodeS algorithm uses quad splits to divide significant sets into smaller sets and calls itself on each offspring. Furthermore, the ProcessI algorithm uses octree partitioning to create new sets of type $\mathcal{S}$, after which it calls the ProcessS algorithm for each of them, which in turn relies on the recursive CodeS algorithm. These recursive processes continue until all newly significant coefficients have been isolated through quad-tree partitioning for a particular bit plane. Finally, the previously found significant coefficients are refined and the process restarts for the next bit plane.

In SPECK, the decoding algorithm is nearly identical to the encoding algorithm, but with send actions replaced by receive actions. When moving a coefficient to the LSP (newly significant), the decoder stores its value as 1.5×2^n, the mid-range of the bit plane where it is significant. During refinement of these sig-

nificant coefficients, this behavior is repeated for lower bit planes.

SPECK encoding example

To clarify the operation of SPECK and its lists, we present a small example using the same set of 4×4 coefficients that we used for the example of SPIHT. Table 2.9(a) shows the coefficient region, and Table 2.9(b) the contents of the lists and sets used by SPECK at the start of the algorithm. In this table and the following section, the notation proposed by Pearlman is used:

- $S^k(i,j)$ denotes a $2^k \times 2^k$ set with (i,j) being the top-left corner coordinate for points or sets,
- $(i,j)\mathbf{k}$ denotes a $2^k \times 2^k$ set with (i,j) being the top-left corner coordinate in the LIS and LSP,
- (i,j) in the LSP always refers to single points.

	0	1	2	3
0	29	15	6	-5
1	12	10	-4	9
2	-5	2	1	-1
3	-6	3	-1	0

(a)

List/Set	Contents
LIS	{ initial set $\mathcal{S}$ }
LSP	{ empty }
$\mathcal{S}$	{ $S^0(0,0)$ }
$\mathcal{I}$	{ remaining 15 values }

(b)

Table 2.9: Example of the SPECK encoding process: (a) matrix of input coefficients, (b) SPECK lists and their contents at the start of the algorithm.

↷ *Initialization*
The number of bit planes n is equal to $n = \lfloor log_2(29) \rfloor = 4$, which is transmitted to the decoder. The lists and sets $\mathcal{S}$ and $\mathcal{I}$ are initialized to the state shown in Table 2.9(b).

↻ *First sorting pass*
At this point, $n = 4$ and the significance threshold $T = 2^n = 16$. First the LIS is processed, which only contains one set $\mathcal{S} = (0,0)$. Since $S_4(\mathcal{S}) = 1$, the set is considered significant and for this a '1' is transmitted. The set only contains one coefficient thus size$(\mathcal{S}) = 1$ and the sign of the coefficient is transmitted, being '0'. The coefficient is added to the LSP. Since $\mathcal{S} \in$ LIS, it is removed from this list. Now that the LIS is completely processed, set $\mathcal{I}$ is processed. Parameter $S_4(\mathcal{I}) = 0$, which is insignificant, for which a '0' is transmitted.

↻ *First refinement pass*
No refinement pass is performed during the first pass, as the LSP is empty at the start.

↻ *Second sorting pass*
Now $n = 3$ and the significance threshold T is $2^n = 8$. First the LIS is processed and skipped, because it is empty. Then, set $\mathcal{I}$ is processed and $S_3(\mathcal{I}) = 1$, which is significant, for which a '1' is transmitted. Set $\mathcal{I}$ is then partitioned into 3 sets $\mathcal{S}$ and a new $\mathcal{I}$. The sets $\mathcal{S}^1(0,1)$, $\mathcal{S}^1(1,0)$ and $\mathcal{S}^1(1,1)$ are processed similarly as $\mathcal{S}^1(0,0)$ and tested for significance. All three are single significant and positive coefficients and '10' is transmitted for each of them (significance=1,sign=0). The three coefficients are added to the LSP. Again set $\mathcal{I}$ is processed and $S_3(\mathcal{I}) = 1$, for which a '1' is transmitted. Set $\mathcal{I}$ is partitioned into 3 sets $\mathcal{S}$ and a new $\mathcal{I}$. The sets $\mathcal{S}^1(0,2)$, $\mathcal{S}^1(2,0)$ and $\mathcal{S}^1(2,2)$ are then processed sequentially. Set $\mathcal{S}^1(0,2)$ is significant, so that a '1' is transmitted and the set is quad-split into 4 points. The first three coefficients $(0,2)$, $(0,3)$ and $(1,2)$ are insignificant and placed in the LIS and '000' is transmitted. The last coefficient is significant and '10' is sent. Sets $\mathcal{S}^1(2,0)$ and $\mathcal{S}^1(2,2)$ are both insignificant, resulting in bit string '00'. Set $\mathcal{I}$ is empty, so that sorting is finished for this bit plane.

↻ *Second refinement pass*
The only location that was already in the LSP at the start of the second pass is $(0,0)$. The fourth bit ($n = 3$) from the Least Significant Bit (LSB) of $c_{0,0}$ is transmitted, and since $|c_{0,0}| = 29 =$`0b11101`, the underlined bit of this string is transmitted, which is a '1'.

→ *Status*
During the first and second pass, '10011010101100010001' was transmitted and the lists now contain the following elements.

LIS	{ $(0,2)$**0**, $(0,3)$**0**, $(1,2)$**0**, $(2,0)$**1**, $(2,2)$**1** }
LSP	{ $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, $(1,3)$ }
$\mathcal{I}$	{ empty }

↻ *Third sorting pass*
Now $n = 2$ and $T = 4$. For process LIS, coefficients $(0,2)$**0**, $(0,3)$**0**, $(1,2)$**0** are significant, move → LSP and transmit '10', '11' and '11'. Quad-split set $(2,0)$**1**, transmit '1' and process its offspring: $(2,0)$, $(2,1)$, $(3,0)$ and $(3,1)$. Coefficient $(2,0)$ is significant → LSP, transmit '11', $(2,1)$ insignificant → LIS, transmit '0',

$(3,0)$ significant $\rightarrow$ LSP, transmit '11', $(3,1)$ insignificant $\rightarrow$ LIS, transmit '0'. Set $(2,2)\mathbf{1}$ is still insignificant, transmit '0'.

$\circlearrowleft$ *Third refinement pass*
Transmit the 3rd bit ($n = 2$) from the LSB of locations $(0,0)$ ($29 = \texttt{0b11}\underline{\texttt{1}}\texttt{01}$), $(0,1)$ ($15 = \texttt{0b01}\underline{\texttt{1}}\texttt{11}$), $(1,0)$ ($12 = \texttt{0b01}\underline{\texttt{1}}\texttt{00}$), $(1,1)$ ($10 = \texttt{0b01}\underline{\texttt{0}}\texttt{10}$), $(1,3)$ ($9 = \texttt{0b01}\underline{\texttt{0}}\texttt{01}$), which cascades to '11100'.

$\rightarrow$ *Status*
During the third pass, '1011111110110011100' was transmitted and the lists now contain the following elements.

LIS	{ $(2,1)\mathbf{0}$, $(3,1)\mathbf{0}$, $(2,2)\mathbf{1}$ }
LSP	{ $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, $(1,3)$, $(0,2)$, $(0,3)$, $(1,2)$, $(2,0)$, $(3,0)$ }
$\mathcal{I}$	{ empty }

This process continues until $n = 0$ and all coefficients are processed. Using the created bitstream, we will give an example of the decoding process in SPECK and compare the intermediate results to SPIHT.

SPECK decoding example

Identical to SPIHT, the decoding process follows the exact execution path of the encoder, so that it has the same data in its LIS and LSP lists. To visualize the decoding of the SPECK bitstream by the decoder, the contents of the output are given in Table 2.10 for the same moments in time as the encoder example above. Table 2.11 gives the results of the SPECK encoding/decoding process after decoding 39 bits. When comparing the tables with those from SPIHT (Table 2.6 and Table 2.7), it is observed that the decoded coefficients are identical, but SPECK accomplishes this transmission with 3 bits less and without relying on inter-band correlations.

Furthermore, it can be noticed that during the third pass, SPIHT and SPECK create an identical bitstream. Even though the algorithms use different partitioning methods, they both converge to a state where the coefficients are processed in the same order. For more complex coefficient fields, this occasional identical order will not hold, but the example does nicely illustrate similarities between the underlying algorithms. Furthermore, the example shows how they exploit the hierarchical nature of the wavelet transform and the associated correlations between wavelet coefficients.

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(a)

24	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(b)

24	12	0	0
12	12	0	12
0	0	0	0
0	0	0	0

(c)

28	12	0	0
12	12	0	12
0	0	0	0
0	0	0	0

(d)

28	12	6	-6
12	12	-6	12
-6	0	0	0
-6	0	0	0

(e)

30	14	6	-6
14	10	-6	10
-6	0	0	0
-6	0	0	0

(f)

Table 2.10: SPECK decoded output after: (a) initialization, (b) 1st sorting pass, (c) 2nd sorting pass, (d) 2nd refinement pass, (e) 3rd sorting pass, (f) 3rd refinement pass.

29	15	6	-5
12	10	-4	9
-5	2	1	-1
-6	3	-1	0

(a)

30	14	6	-6
14	10	-6	10
-6	0	0	0
-6	0	0	0

(b)

1	-1	0	-1
2	0	-2	1
-1	2	-1	1
0	3	1	0

(c)

Table 2.11: Results of the SPECK encoding/decoding process: (a) input, (b) output after decoding 39 bits, (c) difference between input and output.

2.6 Conclusion

This chapter has introduced various aspects of Scalable Video Coding, starting with the need for video compression and providing a short history of current image and video coding, discussing both traditional systems based on the DCT transform and novel DWT-based coding.

To further elaborate on DWT-based coding, we have used a typical scalable wavelet video coding structure as a guideline in this chapter. We have then introduced common wavelet transforms and algorithms from 1D to multi-level 2D,

after which two state-of-the art wavelet coefficient encoding algorithms have been discussed in significant detail: SPIHT (Set Partitioning in Hierarchical Trees) and SPECK (Set Partition Embedded bloCK). By providing a detailed example for each of them, their operation was clearly illustrated.

The algorithm presentations provide a solid basis understanding of the upcoming chapters in this thesis, which focus on our main contributions. Next, in Chapter 3, we will investigate the complexity of the SPIHT and SPECK codec discussed in this chapter. Once the bottlenecks of these algorithms are known, we propose several enhancements to the SPECK codec, in order to simultaneously improve its processing speed, as well as its scalability and coding performance for fast integer-based wavelets.

CHAPTER 

Hardware-efficient scalable wavelet coefficient coding

There are 10 types of people in the world:
Those who understand binary, and those who don't.

UNKNOWN

Abstract

This chapter concentrates on the algorithms for efficient encoding of wavelet coefficients in the area of image coding. It starts by investigating the complexity of the SPIHT and SPECK codecs, after which we propose a hardware-efficient scalable wavelet coefficient coder based on SPECK, named TSSP (Two-Stage SPECK). The proposed codec creates an output stream that is identical to SPECK. Then several enhancements are discussed that significantly improve the efficient implementation of the algorithm as an embedded system. To this end, processing is split into two stages, one data-dependent and one data-independent stage. A highly efficient buffer eliminates the need for dynamic lists and processing in the second stage is performed for all bit planes in parallel. A further advantage is that parallel processing is also facilitated at any level in the hierarchical tree structure. Furthermore, we propose two enhancements. First, the Highly Scalable (HS) extension allows parsing of the bitstream without payload decoding, thereby creating a bitstream of any desired quality, resolution and bitstream order. Second, the 5/3 Energy Correction (53EC) extension retains the perfect reconstruction feature of the 5/3 integer wavelet, while significantly improving lossy coding performance.

3.1 Introduction

In Chapter 2, recent developments in image and video coding have been discussed and various requirements for surveillance image and video coding have been specified. From these specific requirements, we have concluded that scalable video coding based on wavelets is a good candidate to fulfill all functional requirements, but with the considerable complexity of this type of coding, it is difficult to satisfy the system complexity requirement. Therefore, our contribution in this chapter is based on reconsidering the SPECK coding algorithm already at the algorithmic level and modifying the algorithm such that an excellent starting point for obtaining an efficient implementation of the SPECK coding algorithm is enabled, without sacrificing important properties, such as scalability and compatibility. First, the complexities of the wavelet coefficient codecs from Chapter 2 (EZW, SPIHT and SPECK) are investigated, after which a new hardware-efficient scalable wavelet coefficient coding algorithm is proposed.

In the previous chapter, several well-known wavelet coefficient encoding algorithms were discussed: EZW (Embedded Zerotree Wavelet), SPIHT (Set Partitioning in Hierarchical Trees), SPECK (Set Partition Embedded bloCK) and EBCOT (Embedded Block Coding with Optimal Truncation). SPIHT and SPECK have state-of-the-art coding performance and moderate algorithmic complexity. For this reason, these two algorithms have been presented in significantly more detail by discussing their encoding and decoding algorithms and associated examples for both algorithms. These algorithm discussions form a preparation for the computational complexity discussion that follows in this chapter.

Related work. EZW, SPIHT and SPECK all utilize ordered lists to keep track of significant coefficients during bit-plane passes, which results in significant computational complexity for maintaining these lists. Several solutions have been proposed to eliminate the use of lists in these algorithms. Wen-Kuo Lin *et al.* [55, 56] proposed a listless version of SPIHT, named LZC (Listless Zerotree Coding) without lists and with reduced memory requirements, with a small loss in rate-distortion performance. The same authors later improved LZC using raster tree search in [57]. Wheeler and Pearlman [58] proposed NLS (No List SPIHT), storing significance information in a one-dimensional array, similar to Schelkens *et al.* [59], while Jiangling Guo *et al.* [60, 61] suggested to merge the wavelet filter and SPIHT and process the wavelet tree backwards, going from the high- to the low-frequency components, while generating the bitstream simultaneously.

EZW and SPIHT utilize cross-resolution correlations of wavelet coefficients. Especially for large images, this creates significant problems in cache management,

as the algorithms continuously switch between the resolution bands. This was also confirmed for SPIHT by Taubman during his development of EBCOT in [48]. Creusere [62] modified EZW to process the zerotree in so-called partitions, which fit in the local cache memories of a processor.

Due to the embedded nature of the mentioned wavelet coefficient coders, a single wavelet coefficient is passed multiple times during bit-plane coding. As the coefficients in the image are effectively passed 8 to 12 times, it requires a significant memory bandwidth. Creusere [62] has modified EZW to process several bit planes and resolution layers together, albeit at the cost of losing some scalability and rate-distortion performance. The wavelet coefficient coders from literature are also highly data-dependent during the decision making process, as they sequentially test the significance of pixels at the time of coding. This corresponding decision making in the critical path of processing works out poorly for pipelined architectures, which will have to erase various caches at a regular basis.

This discussion brings us to the detailed problem statement of this chapter. The primary question is to employ a highly-efficient wavelet coefficient coder such as SPIHT or SPECK, while at the same time realizing such an advanced coding algorithm at only a fraction of the original complexity. This involves an algorithm that features the following properties: (1) good data-locality, (2) suitability for parallel implementations and (3) a reduction of data-dependency in the processing chain. We have selected SPECK as a basis for our proposal, as it has good data-locality properties and is suited for parallel implementation because it relies on quadtree partitioning.

In our proposal, we will elaborate on and attempt to optimize the use of data lists for improving the data-dependency properties. Furthermore, the wavelet coefficient processing for multiple bit planes is addressed, which also affects data-locality and the decision making process. These two concepts are then applied to the SPECK algorithm in order to significantly reduce the complexity. This involves a.o. splitting the SPECK codec in two stages, resulting in Two-Stage SPECK (TSSP). Besides the previous details, enhancements are also proposed that allow for a highly scalable bitstream representation and special support for the very fast and efficient 5/3 integer wavelet.

This chapter is organized as follows. In Section 3.2, Two-Stage SPECK (TSSP) and the proposed algorithmic changes alleviating the computational bottlenecks of SPECK, are presented. A detailed functional description of TSSP is provided by Section 3.3. Section 3.4 discusses two extensions to TSSP: (1) a highly scalable mode and (2) an energy-correction mode. The highly scalable mode allows full scalability

in all dimensions without the need to decode any part of the bitstream (3.4). The energy-correction mode significantly improves the rate-distortion performance of the 5/3 integer wavelet (Section 3.4). Experimental results regarding the lossless and lossy coding efficiency with and without the proposed extensions are given in Section 3.5, while Section 3.6 concludes the chapter.

3.2 Two-Stage SPECK (TSSP)

In this section, a new form of SPECK coding is proposed, called Two-Stage SPECK (TSSP), which includes adaptations for more efficient coefficient stream handling and the motivation for these adaptations.

Adaptations to SPECK

We propose three adaptations to SPECK, which improve memory usage and processing speed without changing the final bitstream, so that bitstream compatibility is preserved. The first adaptation is to split the SPECK coder in two stages, enabling cascaded processing of SPECK. This stage splitting is discussed in Section 3.2. In the sequel, this architecture is called Two-Stage SPECK (TSSP). For the second adaptation, an intermediate buffer is introduced between the two stages to store the significance level of all blocks in the image. This intermediate buffer is a key feature of TSSP and is addressed in Section 3.2. The third adaptation of the SPECK coder is to process all bit planes in parallel in Stage 2. This means coding decisions involved with coding individual coefficients are made at once for all bit planes simultaneously, so that a significant coefficient is accessed only once in Stage 2. Section 3.2 addresses this one-time reading benefit in more detail. When these three adaptations are incorporated, the block diagram of TSSP becomes as visualized in Figure 3.1.

Motivation for two-stage processing

The two-stage processing in TSSP enables cascaded processing. The first data-independent stage has a fixed pre-determined access pattern, through which it populates a temporary buffer with information about the significance of quadtree partitioning elements. This is called the Significance Level (SL) buffer, which is depicted at the bottom of Figure 3.1 (further discussed in the next section). The first stage has strong optimization possibilities and it can be easily split in several independent processing areas with identical computing structures and thus equal

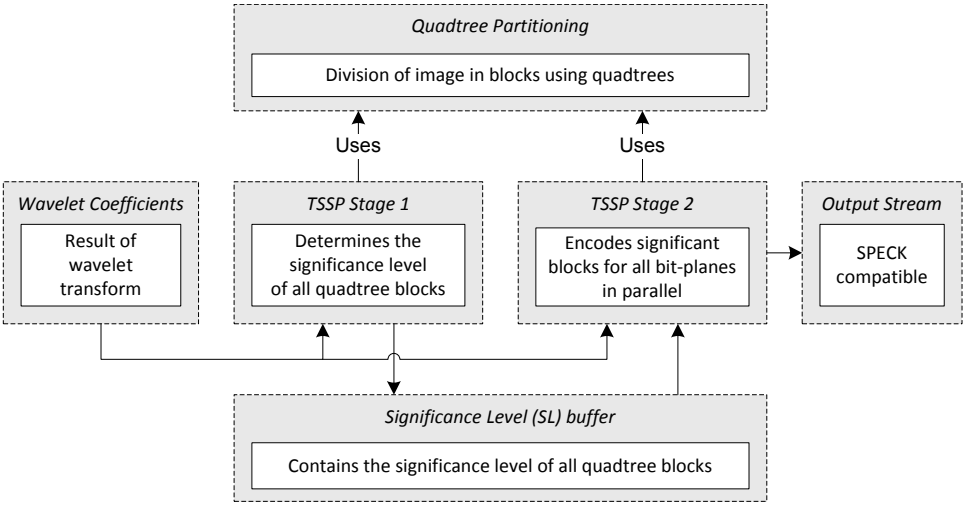


Figure 3.1: Functional blocks of Two-Stage SPECK (TSSP).

computation times, thereby enabling parallel processing on multi-core architectures.

The second data-dependent stage does not need to investigate the significance of individual wavelet coefficients, but utilizes the temporary buffer populated at the first stage. From this buffer, significance information of the quadtree is used to create sorting information for significant coefficient regions and to skip large insignificant coefficient regions. Similar to the first stage, the computing structure can be also split in independent processing areas, but the computation will contain data dependencies, leading to variations in computation time.

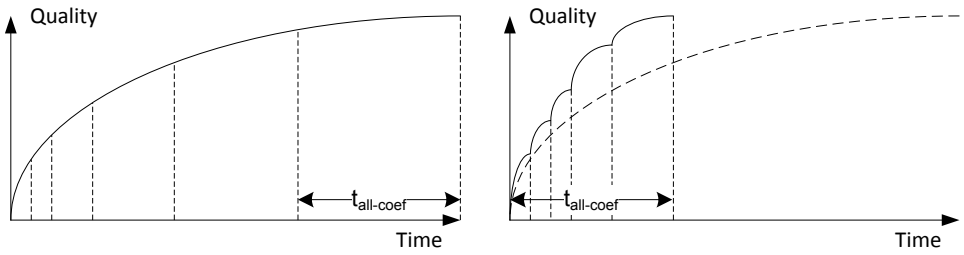
Motivation for intermediate Significance Level (SL) buffer

The intermediate SL buffer is introduced for four reasons. First, it separates the data-independent processing of Stage 1 from the data-dependent processing of Stage 2, transferring the results of Stage 1 to Stage 2. Second, it collects significance information on a per-block basis, for all levels in the quadtree, which is also stored in a specific order. The information in the SL buffer is the sole information needed in Stage 2 for all partitioning decisions. Third, the specific order of storing the significance level information is designed in such a way, that read-access in Stage 2 is always in a forward fashion and insignificant parts can be quickly skipped. Fourth, by only storing block-based significance information, but no pixel

information, the SL buffer remains small enough to be implemented in very fast (second level) cache memory of a processor. For example, for a typical image size of 1280×720 pixels, the SL buffer requires only 85.3 kB of memory space.

Motivation for parallel bit-plane processing

The main reason to process all bit planes in parallel is total computational time, combined with the quality that can be obtained after a certain computational time. When processing bit planes one by one, the output stream is generated in an optimal embedded fashion over time, as seen at the left side of Figure 3.2. However, since most coefficient values are examined multiple times for each bit plane, a large amount of memory bandwidth is used. Since memory bandwidth is one of the bottlenecks in most Digital Signal Processing (DSP) systems, this processing approach is not optimal in terms of processing speed. Therefore, all bit planes are processed in parallel, requiring only the memory transfers equivalent to the last bit-plane pass of the approach when sequential bit-plane processing was used. The time required to retrieve all significant coefficients is equal to the time required to retrieve all coefficients during the last bit-plane pass. This is shown at the left and right side of Figure 3.2 by the parameter $t_{all-coef}$. Because processing is performed progressively in resolution instead of progressively in quality, the quality increase over time is less effective, since less significant bits of a lower resolution are processed prior to more significant bits at higher resolutions. However, when comparing the quality improvement over time, it is clear that the quality after any given processing time, is always equal or better than when sequential bit-plane processing is used.



(a) Bit-plane by bit-plane coding (b) Parallel bit-plane coding

Figure 3.2: Graph of quality vs. processing time for (a) sequential bit-plane coding and (b) parallel bit-plane coding.

3.3 Functional description of TSSP

In this section, TSSP is described in detail, explaining the functional blocks of TSSP of Figure 3.1 in dedicated sections. First, quadtree partitioning is discussed, which is designed to facilitate encoding of non power-of-two image sizes (Section 3.3). Then the data-independent Stage 1 and the data-dependent Stage 2 processing of TSSP are presented separately, in Sections 3.3 and Section 3.3, respectively.

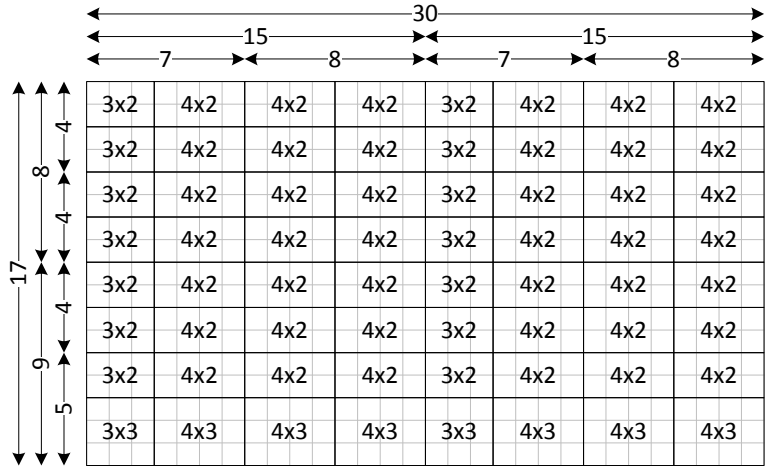
Quadtree partitioning of images

To facilitate arbitrary image sizes, special care is taken in TSSP to provide quadtree partitioning for non power-of-two image sizes. The depth of the quadtree $N_{QTdepth}$ is defined by the number of quadtree partitioning steps that can be made until the remaining areas of coefficients have a width or height of 2 or 3 coefficients. This can be calculated iteratively using the following pseudo code in Algorithm 3.1, with $QTdepth$ denoting the depth of the quadtree.

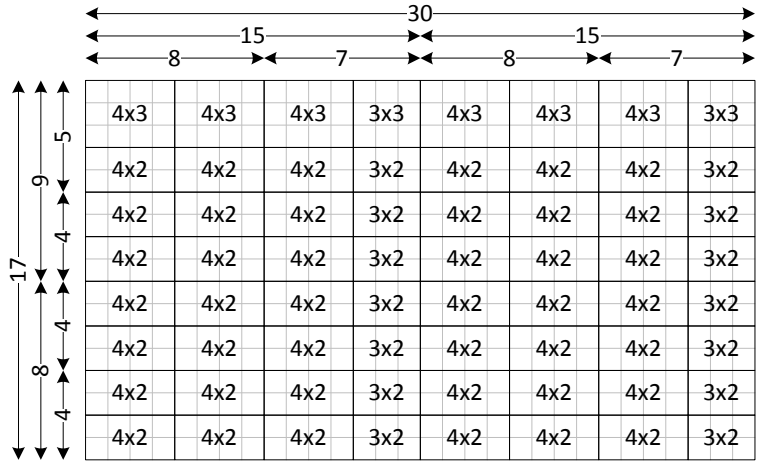
Algorithm 3.1 Pseudo code to calculate quadtree depth.

```
1: procedure QTDEPTH(width, height)          ▷ Width and height of input image
2:     QTdepth ← 0
3:     while width ≥ 4 and height ≥ 4 do      ▷ Perform quadtree partitioning
4:         width ← ⌊width/2⌋
5:         height ← ⌊height/2⌋
6:         QTdepth ← QTdepth + 1
7:     end while
8:     return QTdepth                        ▷ Reached leaf of quadtree
9: end procedure
```

At a certain point in the partitioning process, when the width or height of blocks becomes odd, they cannot be split anymore into 4 blocks of equal size and a decision needs to be made on how the block is further partitioned. TSSP supports two quadtree partitioning methods. First, a partition called 'top-left floor', which means that the size of the top-left partition is determined with the floor operator ($\lfloor x \rfloor$). The second partition is called 'top-left ceiling', which is based on the ceiling operator ($\lceil x \rceil$). These two partitioning methods are visualized in Figure 3.3 for an image with a resolution of 1920×1088 pixels, for which the quadtree partitioning becomes irregular at level 6, where the block size is 30×17 pixels. For the wavelet transform, the same rounding of the size of low- and high-pass bands should be used. The top-left floor partitioning is recommended for regular use, so that the low-pass band of the wavelet will have the smallest size, thereby improving compression efficiency. However, for shape-adaptive wavelets, it is desirable



(a) Quadtree partitioning using the floor operator for assigning top-left partition sizes



(b) Quadtree partitioning using the ceiling operator for assigning top-left partition sizes

Figure 3.3: Size-invariant quadtree partition for a 30×17 pixel region, based on (a) floor and (b) ceiling functions for calculation of the top-left partition size. The displayed region is representative for an image with 1920×1088 pixels, for which the quadtree becomes irregular after the 6th decomposition step at the level of 30×17 pixel regions.

to generate a larger low-pass band to preserve DC information, so that the top-left ceiling partitioning is preferred.

The number of elements in the quadtree N_{QTelem} with l quadtree levels can be calculated by the recursive Equation (3.1), which is defined by

$$N_{QTelem}(l) = 4 \cdot N_{QTelem}(l-1) + 1, \tag{3.1}$$

with $N_{QTelem}(0) = 1$ and $l > 0$. To give the reader an understanding of the sizes of the quadtree, for an image of 1280×720 pixels, the quadtree depth is 8 and the quadtree size is 87,381 elements. This topic will be discussed in more depth in Chapter 6, which covers mapping TSSP on an embedded architecture.

Stage 1: decisionless pre-processing

In the first stage of the TSSP, the Significance Level (SL) of every node of the quadtree is stored in a buffer. The buffer is organized such that the first element contains the SL of the whole image, the second element the SL of the top-left quadrant, etc. Since the SL of a block is equal to the ceiling of the base-2 logarithm of the maximum of its four quadrants, this buffer is generated from the quadtree leafs up to the root of the tree. Figure 3.5 shows the recursive process used in Stage 1, which is presented in pseudo-code by Algorithm 3.2. The first block that initiates processing consists of the whole image. The quadtree partitioning discussed in the previous section, then recursively divides the image into smaller blocks, until a block cannot be split anymore. At this point, the maximum significance level is calculated of all coefficients combined and written to the SL buffer. Also, when all four quadrants of a split block have been processed, their four SL values are combined into the joined SL value representing the complete block, which is also stored in the SL buffer.

By starting the calculations at the bottom-right of the image and progressing in a reversed Morton-order, the SL buffer is generated backwards. This ordering process is visualized in Figure 3.4 for a quadtree of depth 3 with 85 elements. The SL of the bottom-right four leafs is calculated first, after which the SL of the node can be calculated, indicated in the figure by white dots. Once all four nodes at that level are calculated, the SL of the parent node can be calculated, indicated in the figure by the grey dots. Finally, the SL of the whole image can be calculated, referring to the black dot in the middle of the figure. This quadtree consists of 64 leafs, 16 level-2 nodes (white dots), 4 level-1 nodes (grey dots) and 1 level-0 node (black dot), leading to a total of $64 + 16 + 4 + 1 = 85$ quadtree elements.

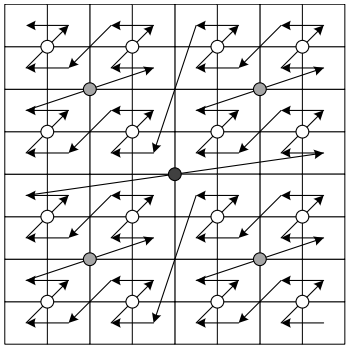


Figure 3.4: Reversed Morton-order scanning in Stage 1 of the TSSP.

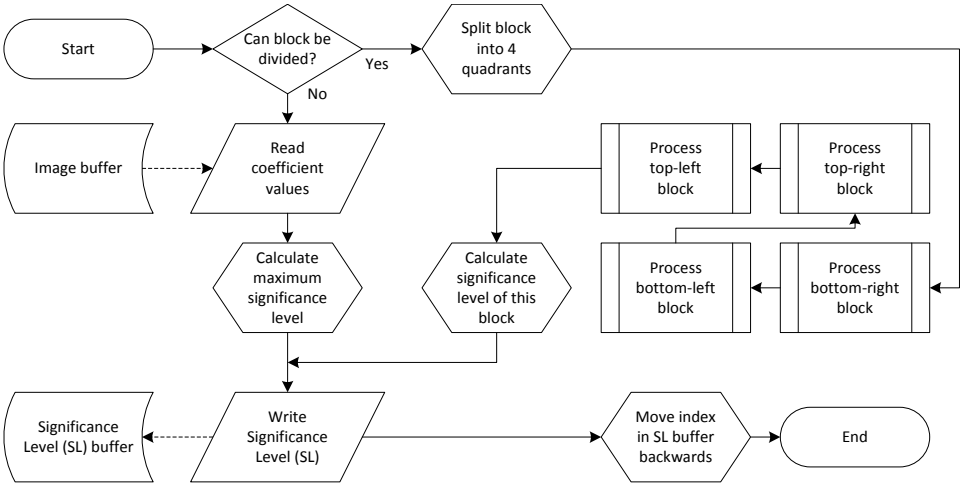


Figure 3.5: Flow chart of block processing in Stage 1 of TSSP.

Stage 2: block and coefficient processing

In the second stage of the TSSP, the SL buffer created at the first stage is used to make data-dependent coding decisions. Based on the SL level of a node in the quadtree, it is determined if this node is significant and should be partitioned further, or if it should be skipped. Individual coefficients do not need to be observed, since only information from the SL buffer is used for decision making.

The order in the SL buffer is designed in such a way that the reading is always in the forward direction. As long as the blocks and their partitions are significant, the next value from the SL buffer is read. Once an insignificant block is encoun-

Algorithm 3.2 TSSP Stage 1: recursively fills significance level backwards

```
1: if the block can be divided then
2:     Split block into 4 quadrants
3:     Process bottom-right block (Recursive call to this algorithm)
4:     Process bottom-left block (Recursive call to this algorithm)
5:     Process top-right block (Recursive call to this algorithm)
6:     Process top-left block (Recursive call to this algorithm)
7:     Calculate significance level of this block from offspring
8: else
9:     Read coefficient values from image buffer
10:    Calculate maximum significance level
11: end if
12: Write block Significance Level (SL) to buffer
13: Move index in SL buffer backward
14: Return block significance level
```

tered, the whole quadtree below this block is insignificant by definition and will be skipped. The values in the SL buffer that represent this insignificant part of the quadtree can be skipped as well. The number of SL values to skip (Δ_{index}) can be calculated directly from the current level in the quadtree $N_{QTlevel}$ by

$$\Delta_{index} = \sum_{l=0}^{N_{QTlevel}} (4^l) - 1 = \frac{4^{N_{QTlevel}+1} - 1}{3} - 1. \tag{3.2}$$

Sorting and refinement data is generated for all bit planes in parallel for each block and coefficient coding step. When blocks are split or skipped, sorting data is generated and when the end of the quadtree is reached, individual coefficients are encoded, generating sorting and refinement data. Since data is generated for multiple bit planes at once, temporary sorting and refinement buffers for each of the bit planes are utilized. These buffers are cascaded at the end of the coding stage to create the final progressive-quality or progressive-resolution bitstream.

The encoding quality is adjusted by the minimum level of significance of the wavelet coefficients, indicated by the Bit-Plane Reduction (BPR) parameter, implementing coarse-grain quality control. Any wavelet coefficient with an absolute value smaller than 2^{BPR} is considered insignificant. The encoding process of blocks and individual coefficients is described in the following sections.

Block encoding process

Blocks are encoded by reading their SL from the buffer created in Stage 1. Figure 3.6 shows the process of block encoding and Algorithm 3.3 describes the process in pseudo-code. Following the figure and the algorithm, it can be observed

that if a block is considered significant ($SL \geq BPR$), a '1' is written to the sorting buffer for the bit plane that has become significant and a '0' in the sorting buffers for each bit plane above that, to indicate their insignificance at that bit-plane level. Afterwards, the block is split into 4 quadrants according to the TSSP partitioning scheme and the process repeats itself, using the next SL value from the SL buffer. If the block is considered insignificant ($SL < BPR$), a '0' is written to the sorting buffer for all bit planes and the underlying tree is skipped. The index skip of the SL buffer is calculated based on the current quadtree depth using Equation (3.2).

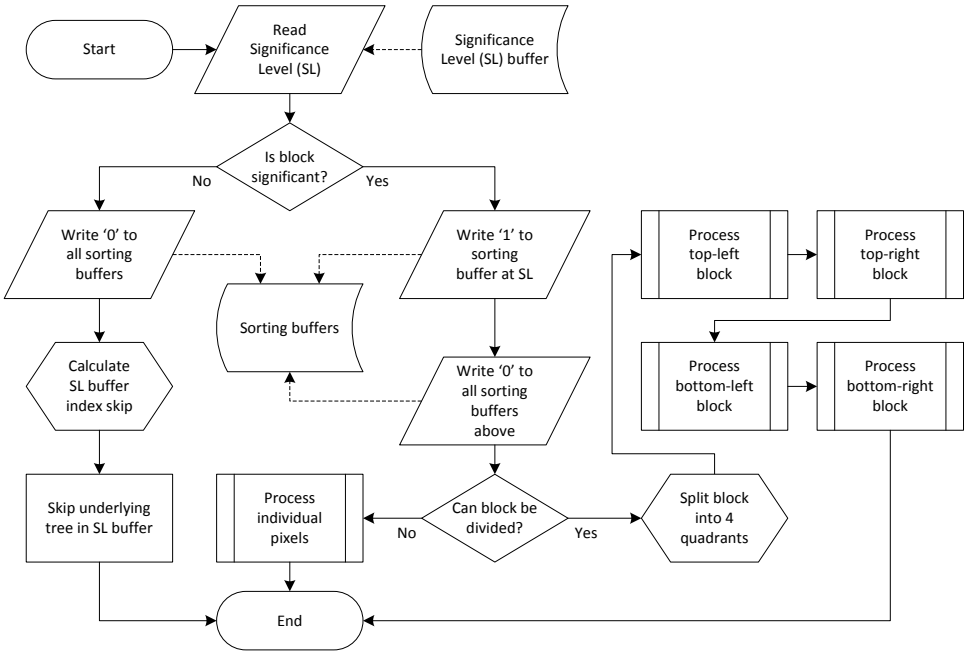


Figure 3.6: Flow chart of the block processing in Stage 2 of TSSP.

Coefficient encoding process

Individual coefficients are encoded once a significant block cannot be split into 4 quadrants, which occurs at the leafs of the quadtree, e.g. the 3×2 top-left block in Figure 3.3(a). The coefficients in the block are encoded row-by-row, starting at the top-left corner. Figure 3.7 shows the process of single coefficient encoding and Algorithm 3.4 describes the process in pseudo-code. Following the figure and the algorithm, it can be observed that first the coefficient value is retrieved from the image buffer. It should be noted that the sorting and refinement buffers are

Algorithm 3.3 TSSP Stage 2 block encoding process

```
1: Read the SL of current block from the SL buffer created in Stage 1.
2: if the block is considered significant (SL ≥ BPR) then
3:     Write a '1' to the sorting buffer for the bit plane that has become significant
4:     for each of the sorting buffers above do
5:         Write a '0', indicating the block is not yet significant for those levels
6:     end for
7:     if the block is larger than the minimum leaf size in the quadtree then
8:         Split the block into 4 quadrants
9:         Process top-left block, top-right, bottom-left and bottom-right block
10:    else
11:        Process individual coefficients using Algorithm 3.4
12:    end if
13: else
14:    Write a '0' to the sorting buffer for all bit planes
15:    Calculate the index skip for the SL buffer using Equation (3.2)
16:    Skip underlying tree by moving the index in the SL buffer
17: end if
```

layered similar to the bit-plane layers which can be best understood at Figure 3.7. The numbering in this figure is such that the LSBs refer to the lowest level numbers and the MSBs to the highest.

If the pixel is not significant ($SL \leq BPR$), '0's are written to all layers of the corresponding sorting buffers and processing is finished. If the pixel is considered significant ($SL \geq BPR$), a '1' is written to the sorting buffer layer, associated with the bit plane at which the pixel is first significant, followed by a '1' if the coefficient value is positive and a '0' in case of a negative coefficient value. Next, a '0' is written in the sorting buffer layers with a higher layer index, to indicate their insignificance at that bit-plane level. Then '0's are written to all sorting buffers with a higher layer index and the refinement bits are written to all refinement buffers with a lower layer index.

Example

The individual coefficient encoding process is explained by providing an encoding example for a coefficient of value 12,289. This coefficient coding process is visualized in Figure 3.8. The value of 12,289 has its first significant bit at bit-plane level 13. Therefore, a '1' is written to the sorting buffer associated with bit plane 13, followed by a '0' indicating the positive sign. For each of the sorting buffers with a higher layer index, a '0' is written, indicating the coefficient is not yet significant for those levels. For bit-plane levels $0 \ldots 12$, the refinement bits are written to the corresponding refinement buffers.

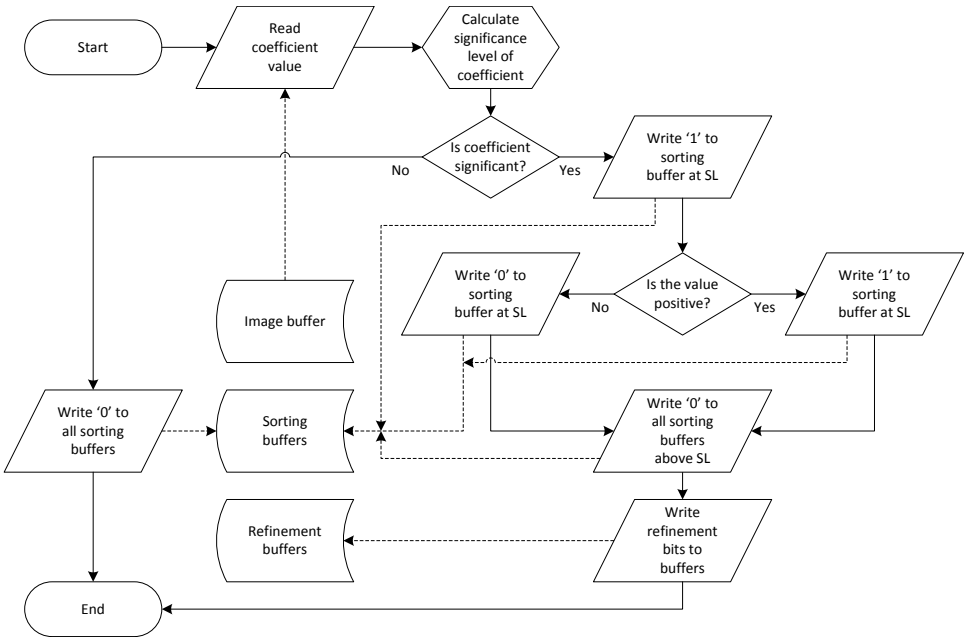


Figure 3.7: Flow chart of the individual coefficient processing in Stage 2 of TSSP.

After all parts of a wavelet decomposition level are coded, the separate sorting and refinement buffers are re-ordered. They can be simply cascaded to generate a progressive-quality bitstream identical to the original SPECK bitstream, or they can be segmented in separate data blocks for each resolution and bit plane to facilitate highly scalable coding. This highly scalable concept will be further elaborated in the following section.

Course-grain quality control

As mentioned in the beginning of this section, the encoding quality in TSSP is adjusted by the Bit-Plane Reduction (BPR) parameter. The BPR parameter reduces the amount of bit planes involved during the block and coefficient encoding process in Stage 2 and allows for an elegant coarse-grain quality control, with minimal computational complexity.

Furthermore, in general, the computational complexity of TSSP and bit-plane based coders can be reduced by moving this coarse-grain quality control to earlier stages in the encoding process, such as the integer wavelet transform and Stage 1 of TSSP. This exceeds the scope of this chapter, but is presented in Appendix A for the interested reader. Because it builds on concepts of TSSP described in this

Algorithm 3.4 TSSP Stage 2 individual coefficient encoding process

```
Determine significance level of coefficient
if coefficient is significant then
    Write a '1' to the sorting buffer at SL bit plane
    if sign of coefficient is positive then
        Write a '0' to the same sorting buffer indicating a positive value
    else
        Write a '1' to the same sorting buffer indicating a negative value
    end if
    for each of the sorting buffers above do
        Write a '0', indicating the coefficient is not yet significant for those levels
    end for
    for each of the refinement buffers below do
        Write the refinement bits, extracted from the coefficient value
    end for
else
    Write '0's to all sorting buffers
end if
```

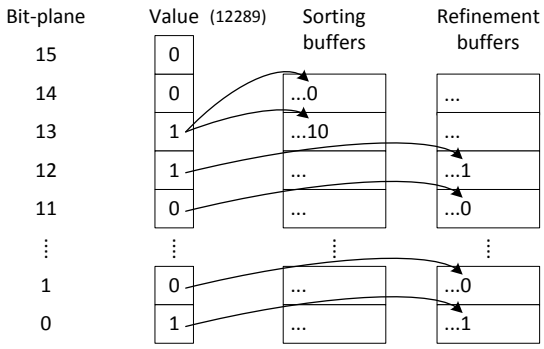


Figure 3.8: Individual coefficient encoding process in Stage 2 of the TSSP for an example `int16` coefficient value of 12,289.

chapter, it is recommended to read the appendix after this chapter.

In summary, Appendix A shows that both methods show a significant and similar reduction in processing time. We have found that for TSSP, bit-plane dropping in Stage 1 is preferred in terms of complexity and processing time, as it does not require any modifications to the wavelet transform. A processing time reduction can be observed for TSSP compared to the standard TSSP, up to 48%. In the scenario of images with 1920×1080 pixels and a typical level of quality control (3 bit planes dropped), an average processing time reduction of 28% is achieved. Furthermore,

it has been found that moving quality control out of the entropy coding into the wavelet transformation is generically applicable to other coding algorithms using bit-plane based quality control, while offering nearly the same processing time reduction, thus leading to a higher processing speed.

TSSP as described up to this point will generate a bitstream identical to SPECK for the same set of wavelet coefficients. The following section presents two extensions to TSSP, which will enhance its scalability and coding performance, albeit losing compatibility with the SPECK bitstream.

3.4 TSSP extensions: deviation from the SPECK bitstream

In this section, two extensions to TSSP are discussed. The first extension features a highly scalable mode providing full scalability in all dimensions, without a need to decode any part of the bitstream (Section 3.4). The second extension is an energy-correction mode, which significantly improves the rate-distortion performance of the 5/3 integer wavelet (Section 3.4). With these modifications, we improve the functionality of TSSP over SPECK, but also lose compatibility between the TSSP and SPECK bitstream.

Highly Scalable (HS) mode

The basic TSSP bitstream only provides fine-grain quality scalability by truncation of the bitstream, which retains compatibility of the bitstream for a regular SPECK decoder. To facilitate full scalability in resolution and quality and to enable the use of the TSSP parser and decoder, the highly scalable TSSP bitstream consists of separate data blocks, each with the bitstream of a particular bit plane and resolution level. If this Highly Scalable (HS) mode is used, the TSSP decoder can be constructed using the same principles and benefits as the TSSP encoder, such as parallel processing of separate data blocks.

The standard SPECK stream and the Highly Scalable (HS) TSSP stream are compared in Figure 3.9. The SPECK bitstream only contains some metadata information about the image, such as the width and height of the image, the number of wavelet decompositions and resolution levels, followed by a continuous bitstream. For TSSP with the HS mode, additional header information is included, which contains indication bits about the chosen filters and indication bits for stream alignment. Standard stream alignment is at Byte level, to avoid complicated bit manipulations within a Byte. Additionally, alternative alignment configurations can be indicated in the header as well, e.g. larger word-length alignments to match with the memory bus width, at the cost of a slightly less efficient bitstream.

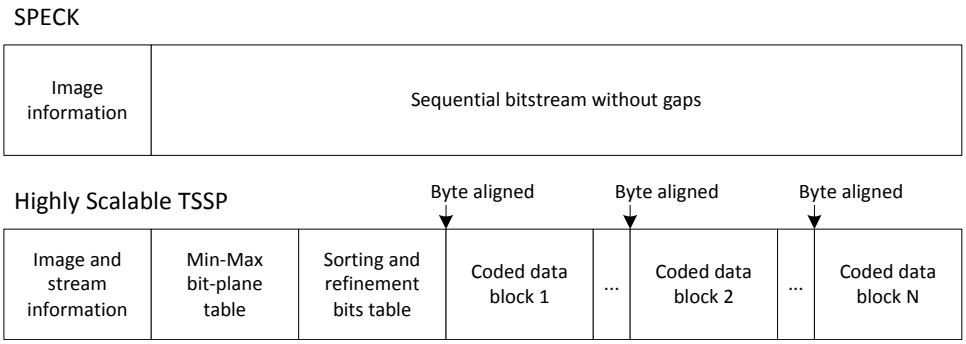
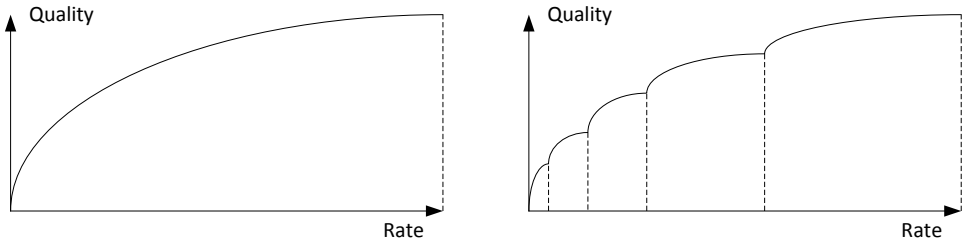


Figure 3.9: Comparison between SPECK and Highly Scalable TSSP bitstream.

Furthermore, two tables are included in the header, the first with the minimum and maximum bit planes for each of the resolution layers. The second table indicates the number of sorting and refinement bits per resolution/bit-plane block. The header is followed by the data blocks, which are aligned to the boundaries specified in the header. So for a Byte-aligned bitstream, each data block starts at a Byte boundary. This may lead to small gaps in between the data blocks, which are filled with 0's. In the original SPECK bitstream, resolution scalability cannot be achieved without decoding the bitstream. The HS-TSSP bitstream allows for scalability without detailed coding operations, which is achieved by simply removing data blocks and updating the header.

The TSSP bitstream can be ordered in a progressive-quality or a progressive-resolution order. For the *progressive-quality order*, data blocks of bit planes of each resolution layer are stored consecutively, followed by the data blocks for all resolution levels of the next bit plane. For the *progressive-resolution order*, data blocks for all bit planes of one resolution layer are stored consecutively, followed by the bit planes of the next resolution layer. These two progressive bitstreams produce different quality vs. rate curves. Figure 3.10 shows the two quality-rate curves (QR curves) for (a) one quality-progressive bitstream and (b) one resolution-progressive bitstream. In sub-figure (b), the vertical dotted lines represent the separation between resolution layers. From Figure 3.10 it can be concluded that it is generally preferable to store the information in a quality-progressive order, since a sudden (unintended) termination of the bitstream will yield the highest quality image up to the point that the information was received.

Scalability is achieved through the use of the TSSP parser, which prunes and reorders the TSSP bitstream at any time after encoding, to create a bitstream with



(a) Quality-progressive bitstream (b) Resolution-progressive bitstream

Figure 3.10: (a) Quality-progressive and (b) resolution-progressive bitstreams decoded at full resolution up to a certain point of reception.

a desired quality, resolution and progression order. This parser only utilizes information from the header and does not need any payload-data decoding from the data packets. As a result, the parser only creates a new header and reorders the bitstream using simple and efficient memory copy operations. In a network environment, data blocks can also be assigned different Quality-Of-Service (QOS) levels, thereby enabling graceful quality/resolution degradation in case of network congestion. Furthermore, by removing the headers and alignment bits, the original SPECK bitstream can be recreated for backwards compatibility.

A detailed description of the highly scalable TSSP bitstream is beyond the scope of this chapter and is therefore specified in Appendix B. It should be noticed that this bitstream description is complete and enables full decoding.

5/3 Energy-Correction mode (53EC)

Here, TSSP is extended with energy correction for the well-known 5/3 integer wavelet. The 2D energy-correction factors derived in Section 2.3 for the 5/3 wavelet are powers of two. Instead of correcting the wavelet coefficients, the bit-plane truncation of the scalable TSSP codec is altered to achieve a similar result. With this codec modification, the original 5/3 integer wavelet can be used, while achieving better lossy coding performance and still supporting lossless coding.

In the regular TSSP, the LH, HL and HH wavelet frequency bands carry equal weight and their bitstreams are combined in a single data block. In the 5/3 energy-correction mode, the data order in the TSSP codec is modified to match the 2D energy correction visualized in Figure 2.4(a). For the proposed 2D energy correction, corrections need to be applied to the LH and HL wavelet frequency bands, which are different from the correction applied to the HH wavelet frequency band.

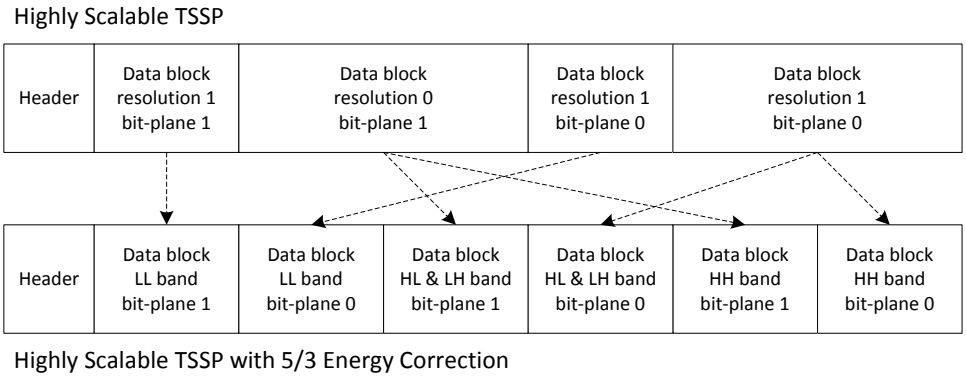


Figure 3.11: Highly Scalable TSSP bitstream without and with 5/3 Energy Correction.

Therefore, in the modified data order, two separate data blocks occur: one for the joined LH–HL wavelet frequency bands and one for the HH wavelet frequency band.

Figure 3.11 shows the difference in stream organization when the 5/3 energy-correction mode is activated for a simple scenario of only 1 iteration of wavelet decomposition and 2 bit planes. From the figure, it can be seen that the data blocks representing the high-pass bands of the wavelet are split into two separate data blocks: one for the joined LH–HL bands and one for the HH band. The stream can now be re-organized to provide better lossy coding performance, by placing the blocks in the correct order, according to their contribution to the image quality.

This principle effectively implements energy correction through stream reorganization and this principle can be expanded to any number of decompositions and bit planes. Figure 3.12(a) shows the data order of the regular scalable TSSP codec for a 3-level dyadic decomposition, with frequency band labels identical to those used in Section 2.3. The encoded data for each frequency band and bit-plane level is visualized by the blocks, while the order in the bitstream is indicated by the arrows between the blocks. The new data order in 5/3 energy-corrected TSSP, based on the additional data blocks and energy correction, is visualized in Figure 3.12(b). It should be noted that even though the energy correction is the same, it is not possible to combine the HH_1 band with the LH_2 and HL_2 bands, as it will destroy resolution scalability. Furthermore, additional descriptive information has to be included in the header, in the form of extra alignment bits for the extra blocks, which will slightly increase the size of the bitstream.

Figure 3.12(c) shows the data order of the 5/3 energy-correction mode, but

(a) TSSP without 5/3 energy correction (b) TSSP with 5/3 energy correction (c) Stretched 5/3 energy correction

Figure 3.12: Visualization of (a) regular TSSP bitstream order with codec-based energy correction and data blocks per band and bit plane: (b) order as stored in bitstream and (c) vertically stretched to clarify amount of 2D energy correction.

now stretched vertically, to visualize the effective energy correction of the blocks with factors $\times 8$, $\times 4$, $\times 2$ and $\times \frac{1}{2}$, identical to the 2D energy correction shown in Figure 2.4(a).

3.5 Experimental results

In this section, both the lossless and lossy coding performance of the proposed TSSP algorithm (Section 3.3) is presented, with and without the highly-scalable and the 5/3 energy-correction extensions from Section 3.4. The experimental results in the following sections are performed on the set of raw test images shown at the end of this chapter in Figure 3.13, with their properties given in Table 3.1. The full-HD images are enlarged slightly by mirroring top and bottom image rows, to allow for 6 levels of dyadic wavelet decompositions required for the SPECK codec implementation. As discussed before, the TSSP codec is capable of encoding arbitrary image sizes.

Table 3.1: Details of the raw images used for experimentation.

Image	Color space / color sub-sampling	Resolution [pixels]	Uncompressed size [kB]
Videoclip	YC_bC_r / 4:2:0	1920×1088	3,060
Bob Marley	YC_bC_r / 4:2:0	1920×1088	3,060
Dude	YC_bC_r / 4:2:0	1920×1088	3,060
Pipe	YC_bC_r / 4:2:0	1920×1088	3,060
Eye	Grayscale	1920×1088	2,040
City still	YC_bC_r / 4:2:0	704×576	594
Crew still	YC_bC_r / 4:2:0	704×576	594
Lena	YC_bC_r / 4:2:0	512×512	384

Evaluation of lossless coding performance

Table 3.2 shows the compressed data sizes for the complete set of test images generated by the TSSP codec with and without the Highly Scalable (HS) mode and for the SPECK codec, where lossless compression factors are represented by the factor indicated between the brackets. The compressed data sizes and compression factors for TSSP without HS mode and SPECK are identical, which is expected as they create identical bitstreams.

The bitstream of TSSP with HS mode includes additional header information and alignment bits, which enables highly scalable parsing of the bitstream without payload-data decoding and facilitates computational parallelism on multi-core architectures for the decoder. From the table, it can be observed that the HS mode increases the data sizes by only 0.1%, which is insignificant when compared to the additional benefits that this mode brings.

Table 3.2: Lossless performance comparison using the 5/3 integer wavelet for TSSP with and without HS (Highly Scalable) mode and SPECK.

Image	SPECK (compr. factor)	TSSP, HS mode off (compr. factor)	TSSP, HS mode on (compr. factor)
Videoclip	1102 kB (2.78×)	1102 kB (2.78×)	1103 kB (2.78×)
Bob Marley	1058 kB (2.89×)	1058 kB (2.89×)	1059 kB (2.89×)
Dude	966 kB (3.17×)	966 kB (3.17×)	967 kB (3.17×)
Pipe	915 kB (3.35×)	915 kB (3.35×)	915 kB (3.34×)
Eye	763 kB (2.67×)	763 kB (2.67×)	764 kB (2.67×)
City still	274 kB (2.17×)	274 kB (2.17×)	275 kB (2.16×)
Crew still	219 kB (2.72×)	219 kB (2.72×)	220 kB (2.71×)
Lena	144 kB (2.67×)	144 kB (2.67×)	145 kB (2.66×)

The lossless-compressed data sizes using the TSSP codec with activated HS mode is evaluated for three cases: (1) using the 5/3 integer wavelet without Energy Correction (EC), (2) using the 5/3 integer wavelet with codec-based EC and (3) applying the 9/7-F integer wavelet. The results for these cases are listed in Table 3.3. It can be observed that the 5/3 integer wavelet without EC has the best lossless coding performance, but the difference with the modified codec with EC is negligible and is only 0.1%. This small difference is due to the small extra overhead caused by the split of the high-pass data into two data blocks, which inevitably increases overhead in header information and alignment bits. The 9/7-F integer wavelet shows a slightly worse lossless performance of 3.5% compared to the 5/3 integer wavelet.

Visual evaluation of lossy coding performance

Now that the lossless coding performance of TSSP has been evaluated, lossy coding performance is investigated, starting with a visual evaluation. On the opposing page of the raw test images of Figure 3.13, the same images are shown in Fig-

Table 3.3: TSSP lossless performance evaluation for integer wavelets with HS (Highly Scalable) mode on, with and without EC (Energy Correction). Compression factors are based on the original bit-rate values, the presented bit rates are rounded.

Image	5/3 wavelet, no EC	5/3 wavelet, with EC	9/7F wavelet
Videoclip	1103 kB (2.78×)	1104 kB (2.77×)	1142 kB (2.68×)
Bob Marley	1059 kB (2.89×)	1060 kB (2.89×)	1099 kB (2.79×)
Dude	967 kB (3.17×)	968 kB (3.16×)	1018 kB (3.01×)
Pipe	915 kB (3.34×)	916 kB (3.34×)	977 kB (3.13×)
Eye	764 kB (2.67×)	764 kB (2.67×)	792 kB (2.58×)
City still	275 kB (2.16×)	276 kB (2.16×)	282 kB (2.11×)
Crew still	220 kB (2.71×)	220 kB (2.70×)	226 kB (2.64×)
Lena	145 kB (2.66×)	145 kB (2.65×)	145 kB (2.65×)

ure 3.14, but decompressed[1] at a Bit-Plane Reduction (BPR) count of 4. This level of compression is significant and ranges from a factor 41 to 99× for HD images and 15 to 41× for SD images. As can be seen from the figure, even at these high compression ratios, the visual quality is still very good. It is also observed that for the same BPR value, the compression factor varies based on the complexity of the image, as the coding algorithm assigns more bits to achieve the desired quality set by the BPR value. The Bob Marley and City images in Figure 3.14(a) and (f), contain many small details and distinct edges, which contain considerable information, requiring a higher bit rate for encoding. On the other side of the spectrum, Dude and Videoclip in Figure 3.14(b) and (d), both have a blurred background due to the limited depth of field of the camera, whereas Crew (Figure 3.14(h)) contains large flat patches without detail. These aspects reduce high-frequency information and lead to a higher compression factor.

Figures 3.15 and 3.16 compare the visual compression artifacts for different levels of BPR for the Pipe and Dude images, respectively. At the left side, the complete image is shown with at the right side a magnified view. At the top, the original image is shown, with decoded images below at gradually increasing levels of compression. Even at high compression ratios, the visual quality is considered very good and the lack of blocking artifacts makes the wavelet transform very suitable for the coding of natural images. For the higher compression ratios, some loss of detail occurs in flat areas, but edges remain well represented. Visually, this is also

[1] All images are compressed only once using HS-TSSP and the 9/7 floating-point wavelet transform and then decompressed at various qualities and resolutions, but always derived from the same compressed bitstream.

perceived as a form of noise reduction.

Finally, Figures 3.17 and 3.18 show the visual quality of reduced-resolution decoding of the Pipe and Dude images, respectively. These reduced-resolution images are all decoded from the same original bitstream. Reduced resolution means here that the original image at the left of the figures is coded at near broadcast to approximately CIF resolution. These images are decoded at resolutions available from the 6-level dyadic wavelet decomposition. Although the compressed files are only several tens of kilobytes, they still offer a very good visual quality. At these resolution levels or somewhat lower, the coding performance decreases because a lower level of hierarchical wavelet decomposition leads to a lower decorrelation of the image. This phenomenon is also observed when comparing the compression factors between HD and SD images, where it was also noticeable that the compression performance increases for higher resolution images, such as Ultra-HD images with a resolution of 3840×2160 pixels.

Analytical evaluation of lossy coding performance

Figures 3.19(a)–(f) show the rate-distortion curves for several images with diverse resolutions and using various lossy and lossless wavelet transforms. The curves are generated with the proposed TSSP codec using the Highly Scalable (HS) mode and the original images are encoded to (near-)lossless quality. The bitstream is then truncated, using the TSSP parser for a range of rate points and decoded using the TSSP decoder. Two non-reversible wavelets are used: the 9/7 floating-point wavelet and the 5/3 integer wavelet with Energy Correction (EC). For the lossless wavelets, the 9/7-F integer wavelet and the 5/3 integer wavelet are employed, the latter with and without codec-based EC.

When comparing different wavelet transforms, for all images, the lossy 9/7 floating-point wavelet yields the best performance, followed by the lossy 5/3 integer wavelet with EC applied to the LL and HH bands. For the lossless wavelets, the 5/3 integer wavelet with EC performed in the codec provides the best results, which are close to the lossy 5/3 integer wavelet, followed by the 9/7-F integer wavelet. From these curves, it is also clearly visible that the 5/3 wavelet without any form of EC is impractical for lossy image coding, as it yields a quality degradation of up to 5 dB.

The performance difference between EC in the wavelet and in the codec is probably explained by two aspects. The first aspect is caused by higher bit rate when using codec-based EC, due to the additionally included header information. However, this does not account fully for the loss in performance. The second aspect is likely resulting from the different incomplete wavelet coefficient rounding at the

decoder, followed by the inverse wavelet transform, with and without internal EC. However, the slight gain in performance for the 5/3 integer wavelet with internal EC comes at the expense of losing perfect reconstruction.

Figures 3.20(a)–(f) show the same curves, but with BPR on the horizontal axis instead of bit rate[2]. From these figures it can be observed that for all wavelet transforms, the PSNR drops at a rate of approximately 3 dB per BPR level. Furthermore, for the different images, the achieved quality in terms of PSNR is similar for comparable levels of BPR. These two features confirm that the BPR value is a good means for control of quality-based coding.

Finally, Figures 3.21(a)–(f) show the rate-distortion curves for a sub-set of 6 images with different resolutions, with and without the HS mode activated. Without the HS mode activated, the TSSP bitstream is identical to the SPECK bitstream, so that their curves are interchangeable. For TSSP with HS, a small quality degradation at the same rate can be observed, or a small rate increase at the same quality, which is explained by the additional header information and the Byte-alignment for all data blocks. For larger images and higher rates, the relative performance difference becomes smaller, as the number of extra alignment bits becomes insignificant compared to the number of bits contained in the larger data blocks. The same applies to the required header bits for each data block.

3.6 Conclusions

In this chapter, the computational bottlenecks of some of the mostly used wavelet coefficient coding algorithms were studied. Although several enhancements have been proposed in literature, none of them fully remove the most prevalent bottlenecks: (1) tracking of significant coefficients throughout bit planes, (2) lack of data localization / possibilities for parallelism, (3) the need for repetitive reading of the same coefficients for several bit planes and (4) data dependency during the coding process.

We have therefore proposed significant algorithmic modifications to the SPECK codec. The first modification consists of splitting SPECK into two stages, resulting in the name Two-Stage SPECK (TSSP). In the data-independent Stage 1, a Significance Level (SL) buffer is created storing the SL of all nodes in the quadtree, which is used in the data-dependent Stage 2 for decision making. Furthermore, in Stage 2 all bit planes are processed in parallel. These algorithmic modifications have the following benefits. A key advantage of the data-independence in Stage 1 is that

[2]The height of the curve in these figures does not indicate which wavelet performs better, as in the previous figure, but merely a different BPR-PSNR relation.

the SL buffer between the two stages acts as an efficient communication medium between the two types of processing, because it features regular access patterns and a fixed computational load. The SL buffer also offers an efficient way to skip insignificant portions of the image and the quadtree in Stage 2, without the need to observe individual coefficient values, thereby improving computational efficiency. Parallel bit-plane processing in Stage 2 utilizes data localization, while eliminating the need for repetitive access of the same coefficient for several bit planes. Furthermore, TSSP is well suited for large-scale parallelism in computation, as it can be split into parts at any quadtree depth, both at Stage 1 and Stage 2.

Two extensions to TSSP have been proposed, through which the TSSP bitstream deviates from the SPECK bitstream: the Highly-Scalable (HS) extension and the 5/3 Energy-Correction (53EC) extension. The HS extension separates the continuous bitstream of SPECK into deliberate data blocks with an index in the header. It also adds an adjustable alignment of the data blocks. This has the following advantages. The data blocks allow parsing of the bitstream without payload decoding, thereby creating a bitstream of any desired quality, resolution and bitstream order, by selectively discarding parts of the bitstream. The adjustable alignment of the data blocks allows TSSP to adapt to the specific ways of accessing data by different architectures[3]. The data blocks also enable full parallel processing in the decoder, at any level in the quadtree. Furthermore, by removing the headers and alignment bits, an original SPECK bitstream can always be re-created for backwards compatibility. The 53EC extension builds upon the HS extension and adds additional data blocks for separate frequency bands, with adjusted bit-plane processing to simulate energy correction. The 53EC extension has the advantage that it retains the perfect reconstruction feature of the 5/3 integer wavelet, while significantly improving lossy coding performance, by up to 5 dB as seen in Figure 3.19, with a negligible performance drop of only 0.1% at lossless coding. TSSP also shows good lossless and lossy compression performance. Even at high compression ratios, the visual quality is considered very good and the lack of blocking artifacts makes the wavelet transform very suitable for the coding of natural images. Furthermore, the BPR parameter provides a robust quality control (3 dB per level), which creates images of similar visual quality and PSNR for the same value of BPR.

The benefits of the TSSP proposed are multifold and not limited to our original scope of embedded surveillance. For clarity, we have divided these benefits in two categories. The two-stage approach facilitates an efficient implementation, which in turn offers benefits such as increased processing speed, power reduc-

[3]Modern architectures include highly optimized data paths for longer word lengths, such as 32 and 64 bits.

tion, heat reduction and battery-life improvements, which is important for embedded cameras and mobile viewing applications. Furthermore, the implementation concept behind TSSP (two-stage processing, integrated energy correction for the 5/3 wavelet within the codec and parallel bit-plane processing) allows for elegant multi-core implementations in general. Second, at the coding side and in a more general view, TSSP provides excellent lossless and lossy performance in synergy with numerous scalability options. As an example, it is possible to utilize quality-driven coding, which can also be applied to other codecs. This flexibility can be employed in many application areas, such as medical imaging, internet distribution and systems based on devices with widely varying specifications.

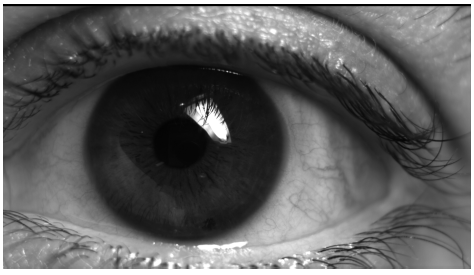
For the next chapter in this thesis, Chapter 4, we will first expand our discussion on scalability and complexity into the time domain to support video processing. It investigates how utilize wavelet filters can be utilized in the temporal domain and how to include motion estimation and compensation in the new setting with temporal processing.

(a) Bob Marley 1920 × 1080 pixels

(b) Dude 1920 × 1080 pixels

(c) Pipe 1920 × 1080 pixels

(d) Videoclip 1920 × 1080 pixels



(e) Eye 1920 × 1080 pixels

(f) City still 704 × 576 pixels

(g) Lena 512 × 512 pixels

(h) Crew still 704 × 576 pixels

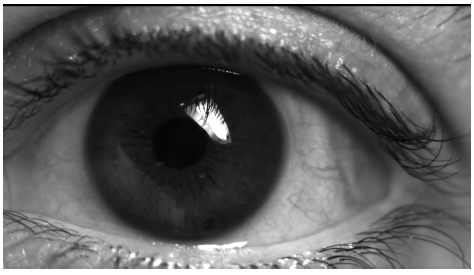
Figure 3.13: Raw images used for experimentation.

(a) Bob Marley (73.96 kB, 41×, 38.56 dB)

(b) Dude (31.34 kB, 98×, 40.01 dB)

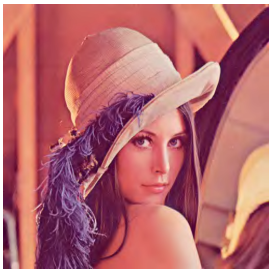
(c) Pipe (63.32 kB, 48×, 38.89 dB)

(d) Videoclip (31.03 kB, 99×, 40.02 dB)



(e) Eye (24.41 kB, 84×, 42.13 dB)

(f) City still (39.74 kB, 15×, 33.96 dB)



(g) Lena (13.44 kB, 29×, 36.39 dB)

(h) Crew still (14.75 kB, 40×, 37.24 dB)

Figure 3.14: Images encoded and then decoded at a BPR of 4 with in between brackets: compressed size in kB, compression factor and Y channel PSNR.

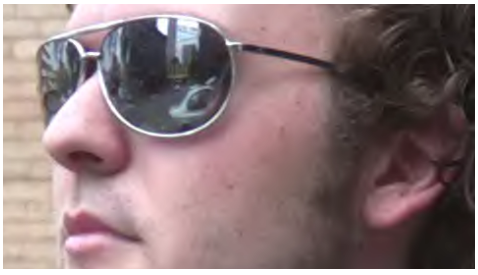
(a) Dude uncompressed (3060 kB)

(b) Dude uncompressed detail

(c) Decompressed with BPR 3 (74.35 kB, 41×)

(d) Decompressed with BPR 3 detail

(e) Decompressed with BPR 4 (31.34 kB, 98×)



(f) Decompressed with BPR 4 detail

(g) Decompressed with BPR 5 (15.01 kB, 202×)

(h) Decompressed with BPR 5 detail

Figure 3.15: Image Dude (1920 × 1088 pixels) decompressed with different BPRs.

(a) Pipe uncompressed (3060 kB)

(b) Pipe uncompressed detail

(c) Decompressed with BPR 3 (120.9 kB, 25×)

(d) Decompressed with BPR 3 detail

(e) Decompressed with BPR 4 (63.32 kB, 48×)

(f) Decompressed with BPR 4 detail

(g) Decompressed with BPR 5 (31.36 kB, 97×)

(h) Decompressed with BPR 5 detail

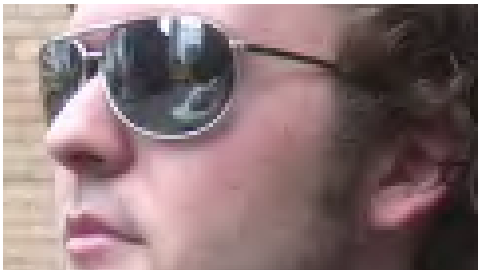
Figure 3.16: Image Pipe (1920 × 1088 pixels) decompressed with different BPRs.

(a) Dude decompressed with BPR 3, full resolution (1920 × 1088, 74.35 kB, 41×)

(b) BPR 3, 1/4 resolution
(960 × 544, 30.28 kB, 100×)

(c) BPR 4, 1/4 resolution detail
(240 × 135 pixels)

(d) BPR 3, 1/16 resolution
(480 × 272, 13.15 kB, 231×)



(e) BPR 5, 1/16 resolution detail
(120 × 67 pixels)

Figure 3.17: Image Dude (1920 × 1088 pixels) decompressed at various resolutions.

(a) Pipe decompressed with BPR 3, full resolution (1920 × 1088, 120.9 kB, 25×)

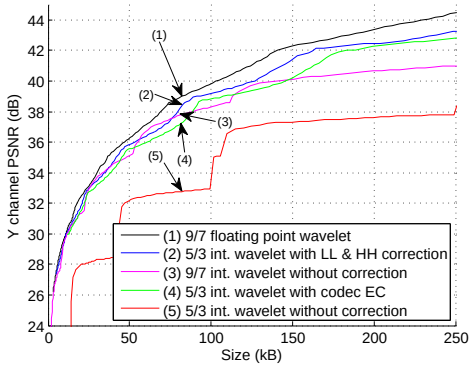
(b) BPR 3, 1/4 resolution
(960 × 544, 62.38 kB, 49×)

(c) BPR 4, 1/4 resolution detail
(240 × 135 pixels)

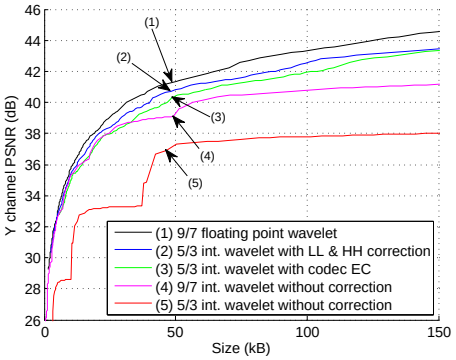
(d) BPR 3, 1/16 resolution
(480 × 272, 24.89 kB, 123×)

(e) BPR 5, 1/16 resolution detail
(120 × 67 pixels)

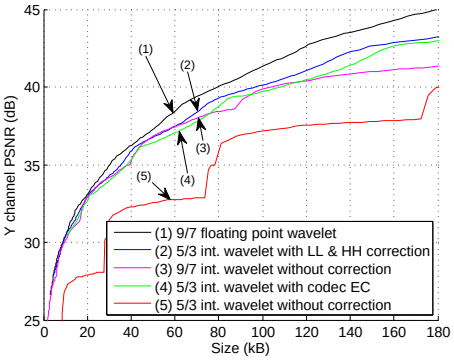
Figure 3.18: Image Pipe (1920 × 1088 pixels) decompressed at various resolutions.



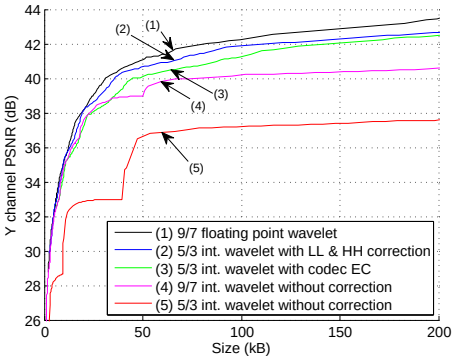
(a) Bob Marley 1920 × 1088 pixels



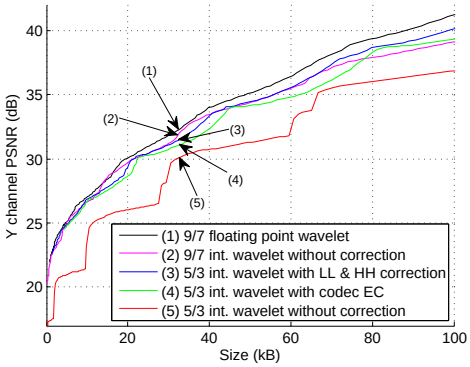
(b) Dude 1920 × 1088 pixels



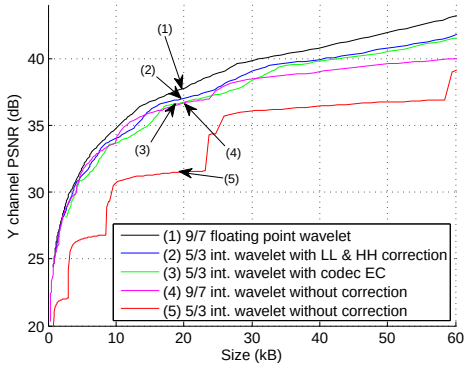
(c) Pipe 1920 × 1088 pixels



(d) Videoclip 1920 × 1088 pixels

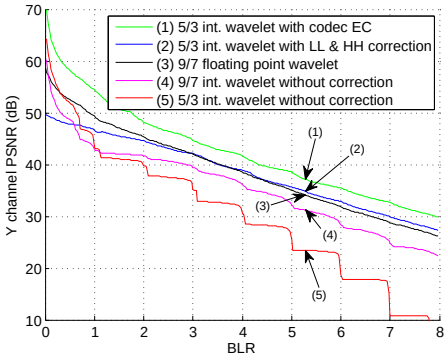


(e) City 704 × 576 pixels

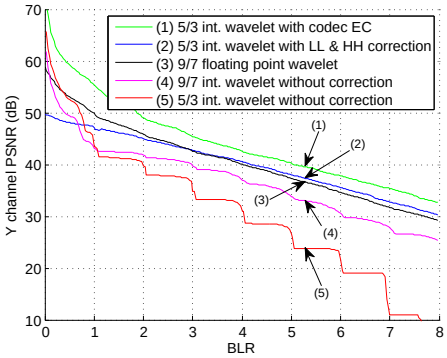


(f) Lena 512 × 512 pixels

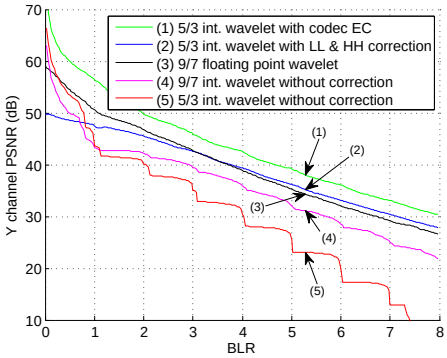
Figure 3.19: Rate-distortion curves for the (a) Bob Marley, (b) Dude, (c) Pipe, (d) Videoclip, (e) City and (f) Lena test images.



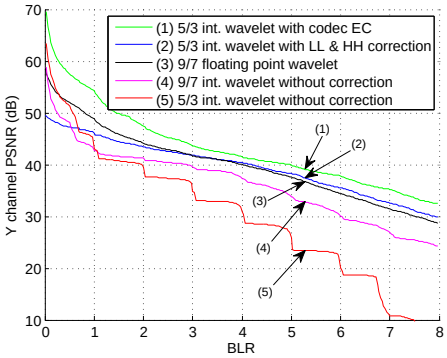
(a) Bob Marley 1920 × 1088 pixels



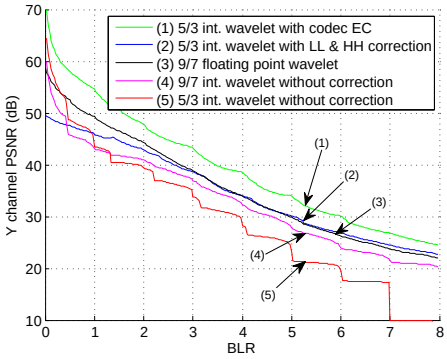
(b) Dude 1920 × 1088 pixels



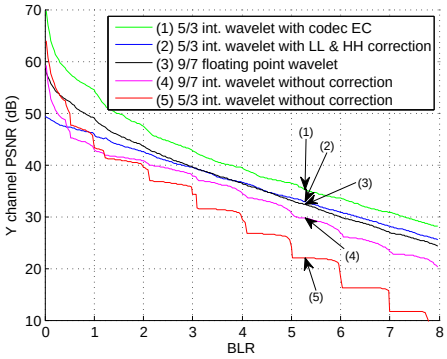
(c) Pipe 1920 × 1088 pixels



(d) Videoclip 1920 × 1088 pixels

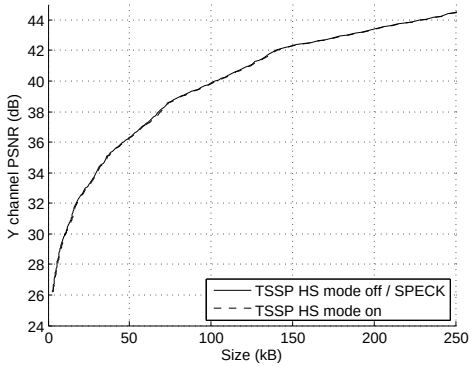


(e) City 704 × 576 pixels

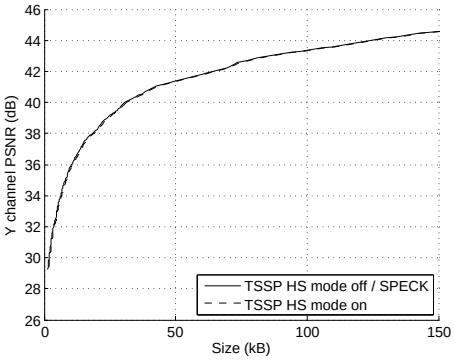


(f) Lena 512 × 512 pixels

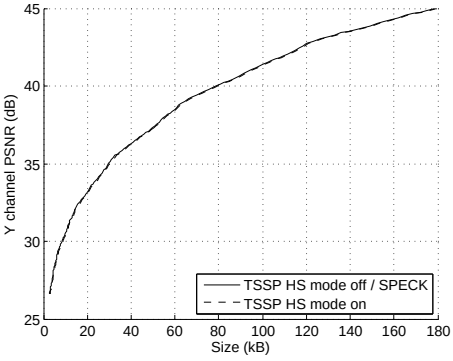
Figure 3.20: BPR-distortion curves for the (a) Bob Marley, (b) Dude, (c) Pipe, (d) Videoclip, (e) City and (f) Lena test images.



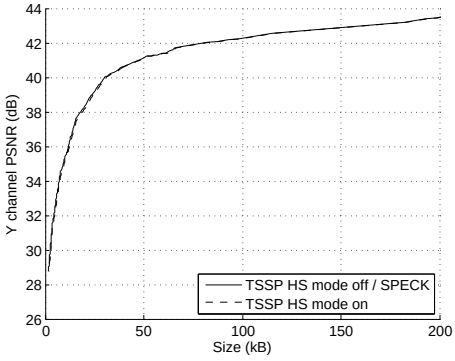
(a) Bob Marley 1920 × 1088 pixels



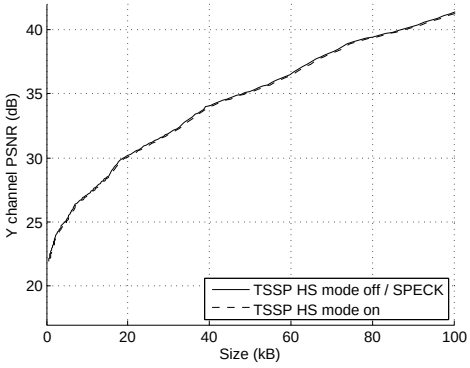
(b) Dude 1920 × 1088 pixels



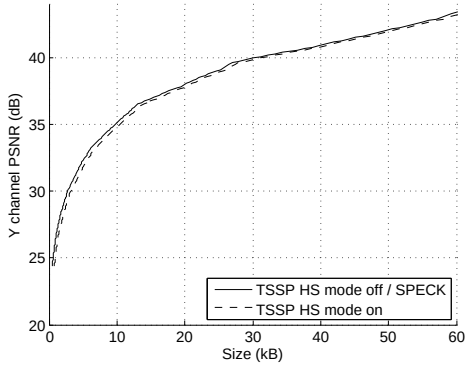
(c) Pipe 1920 × 1088 pixels



(d) Videoclip 1920 × 1088 pixels

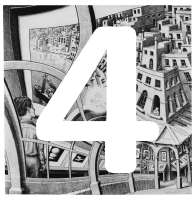


(e) City 704 × 576 pixels



(f) Lena 512 × 512 pixels

Figure 3.21: Comparison between TSSP with Highly Scalable (HS) mode on and off for the (a) Bob Marley, (b) Dude, (c) Pipe, (d) Videoclip, (e) City and (f) Lena test images, using the 9/7 floating point wavelet.

CHAPTER 

Complexity in the temporal domain of scalable video coding

Make everything as simple as possible, but not simpler.

ALBERT EINSTEIN

Abstract

This chapter explores the trade-off between complexity and performance in scalable wavelet coders in the temporal domain. We present a framework to investigate various temporal configurations and after evaluation, adopt a special configuration for video surveillance applications, which features low memory access and low end-to-end delay, while still achieving sufficiently high quality. We also propose two extensions to the framework: Temporal Energy Correction (TEC) and Low-Complexity Encoder Feedback (LCEF). TEC improves energy correction in the temporal lifting tree. A significant reduction in computational complexity is achieved (about 43%), while simultaneously reducing the memory bandwidth even further. LCEF adds the use of a quality-reduced frame at the start of each Group Of Pictures (GOP), which is generated without entropy decoding, but as part of the entropy encoding process. For LCEF, added computational complexity is minimal and decoder modifications are not required. Using this technique, the average quality of the frames within a GOP is improved and quality fluctuations are significantly reduced, which is also expressed by a reduction of the PSNR variance by 30%. LCEF also improves the average quality and combined with the reduced quality fluctuations, leads to a significantly higher perceived quality. With respect to coding quality in general, the coding performance of our proposed SVC at full resolution is close to H.264 SVC (within 1 dB for surveillance type video) and at lower resolutions sufficiently good for the video surveillance application, but with a much lower complexity.

4.1 Introduction

In Chapter 3, scalable image coding based on the wavelet transformation has been investigated. Here, the Two-Stage SPECK (TSSP) coding algorithm has been proposed, which improves the memory usage and processing speed of the SPECK codec, by splitting the processing in two stages, one data-dependent and one data-independent. Furthermore, two extensions to TSSP have been proposed, named HS and 53EC, which provide better scalability and lossy coding performance. However, up to this point, the described properties of TSSP coding have been validated for image coding, but not for video sequences.

This chapter concentrates on extending the scalable coding framework to the temporal domain for video surveillance applications. The complexity implications of variations in the structure of the temporal wavelet filtering are investigated in detail, in order to determine the best structure suited for embedded video surveillance. The analysis will apply several state-of-the-art motion estimation techniques, in contrast with Chapter 5, where a special motion estimator for scalable video coding will be explored.

When considering joint spatio-temporal processing in more detail, the required wavelet filtering is performed explicitly in both the spatial and the temporal domain, also commonly known as 3D subband coding. Coding performance can be further increased by using motion compensation within the temporal wavelet filter. This concept is called Motion Compensated Temporal Filtering (MCTF), which was first proposed by Ohm [63] and later refined by Choi and Woods [64]. Improvements were then made by Secker and Taubman [65] and Pesquet-Popescu and Bottreau [66], which include the utilization of lifting [28] in the temporal wavelet decomposition, similar to the lifting used in the spatial domain as discussed in Section 2.3. Later, a more flexible framework was proposed by Van der Schaar and Turaga [67], known as Unconstrained Motion Compensated Temporal Filtering (UMCTF), which allows for more flexible temporal configurations.

Other research is dedicated to implementation aspects of 3D subband coders. Turaga *et al.* [68, 69] investigated different spatio-temporal decomposition structures for a wide range of Rate-Distortion-Complexity (R-D-C) operating points. Pau and Pesquet-Popescu [70] investigated delay-performance trade-offs in MCTF. Later, Pau, Viéron and Pesquet-Popescu [71] analyzed the encoder and decoder delays and the overall end-to-end delay in the UMCTF framework.

The previous aspects of temporal processing (e.g. lifting, temporal structure, delay) are essential elements in the problem statement of this chapter. The research questions for this chapter emerging from this discussion are how to: (1) analyze the dynamic behavior of 3D subband coders in the multi-dimensional pa-

rameter space and (2) find the balance between computational cost, bandwidth, rate-distortion performance and visual quality. For the development of this chapter, the first question is leading and then gradually evolves to the second question. Based on this exploration, our analysis will lead to the temporal configuration with the best balance for embedded video surveillance, which has strong implications on computational complexity and end-to-end delay.

This chapter is organized as is depicted in Figure 4.1. To further investigate the complexity in the temporal domain, several possible temporal coding structures for wavelet video coding are investigated in Section 4.2. Section 4.3 then explores the variations in temporal configurations that can be be made, followed by a dynamic analysis and complexity estimation of these temporal configurations. Section 4.4 then addresses two extensions to the temporal filtering: (1) for improving the energy balance in the temporal domain and (2) including low-complexity encoder feedback to achieve temporal stability of the perceived quality. Section 4.5 presents the results and shows the trade-offs that can be made between quality and complexity, after which conclusions are given in Section 4.6.

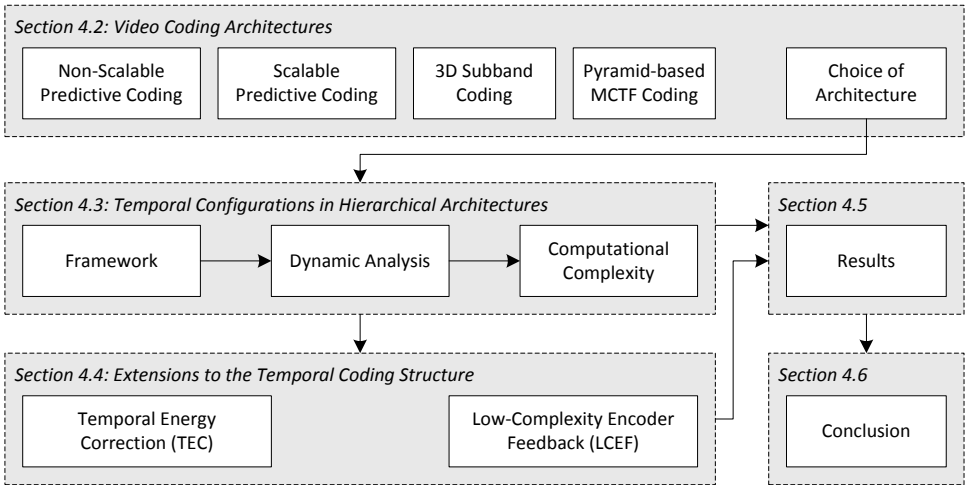


Figure 4.1: Structural diagram of Chapter 4.

4.2 Temporal coding architectures for scalable video coding and their merits

In this section, several possible coding structures for wavelet video coding are investigated to determine their suitability for scalable coding. The two most prominent video compression architectures currently being used for (wavelet) video coding are coarsely categorized in predictive coding and 3D subband coding. This section will discuss the various predictive and subband coding architectures and motivate a choice for further analysis.

Predictive coding systems

Predictive coders are a proven technology in DCT-based video coders (such as the well-known MPEG-1 and MPEG-2 [72, 73] standards) and are easily adapted for wavelet video coding. Figure 4.2 depicts the schematic diagram of a predictive video coder. The first frame is encoded in intraframe coding mode (abbreviated further as intra mode), without temporal processing such as motion compensation. These intra-coded frames or I-frames, are used periodically to initiate the temporal coding process for a group of consecutive frames. The periodic occurrence of I-frames provides protection against temporal error propagation and it enables random access within the compressed video sequence. In the case of wavelet video coding, the image is first transformed using the wavelet decomposition and the wavelet coefficients are quantized. The quantized coefficients are entropy coded and communicated over the channel as a coded I-frame. For the succeeding frames, motion is estimated between the current frame and the decoded version of the previously encoded frame. The decoded frame is then motion compensated to form a prediction of the current frame, after which the difference between this predicted frame and the current video frame is encoded. The quantized coefficients and the motion vectors are encoded and communicated over the channel as a coded predicted frame or coded P-frame.

Modern compression systems not only use uni-directional prediction, but also exploit bidirectional[1] prediction. In such a concept, both forward and backward predicted frames surrounding the actual frame to be coded are used to improve coding efficiency. These B-frames are predicted from I and P-frames and not referenced by other frames. In the latest standards, even B-frames are considered for referencing, such as special profiles of the H.264 standard.

[1] We have adopted the term 'bidirectional' without hyphen in accordance with the coding literature.

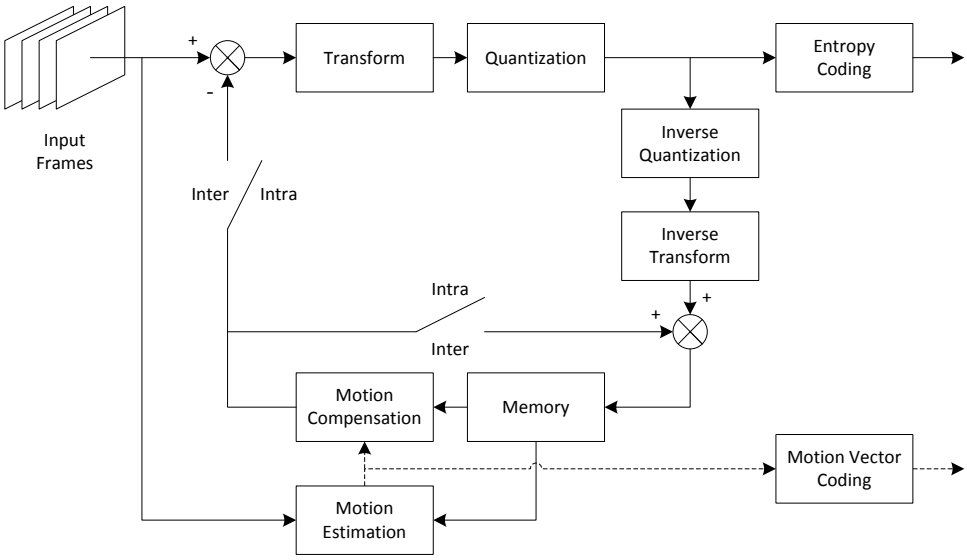


Figure 4.2: Diagram of a non-scalable predictive video encoder (e.g. MPEG-2).

Scalable predictive coding systems

The predictive coder described in the previous section and Figure 4.2 is not scalable. To obtain scalability in predictive coders, a layered approach is commonly employed, in which one or multiple enhancement layers are added to a base layer. A schematic representation of scalable predictive coding is shown in Figure 4.3, where the encoder and decoder blocks represent the predictive coder as described previously. To obtain different resolution levels and frame rates, spatial/temporal up- and down-sampling blocks are introduced. Each layer performs down-sampling of the input frames to the resolution and/or frame rate designated at that particular layer, after which the output of the previous layer is extracted. Traditional non-scalable coding and decoding is performed within each layer and the output signal is up-sampled to the resolution and/or frame rate of the next layer. This is done for a predetermined number of enhancement layers, until full quality is reached. The base layer does not have any inputs from previous layers, while the last enhancement layer neither has down-sampling nor up-sampling. A well-known example of the above structure is the Laplacian pyramid coding system [16].

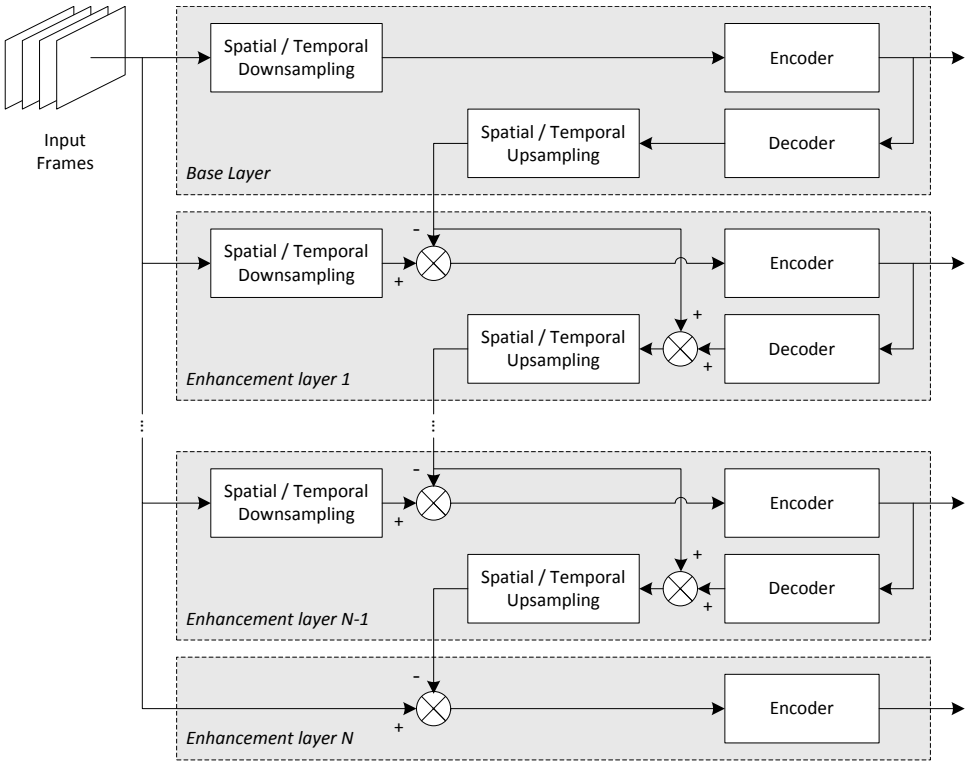


Figure 4.3: Schematic representation of a scalable predictive video encoder.

3D subband coding systems

3D subband video coders have been broadly explored in the past decade, due to their good compression properties with a native support for various forms of scalability. Several forms of 3D subband coders have been proposed by a.o. Ohm [74, 63], Taubman and Secker [75, 65, 76, 77], BeongJoKim *et al.* [78, 79], researchers supervised by Woods [64, 80, 81, 82, 83, 84, 85] and researchers supervised by Pesquet-Popescu and van der Schaar [66, 86, 67, 87, 88, 89, 90, 91, 92]. All these schemes can be generalized to one of the categories abbreviated as t+2D, 2D+t and 2D+t+2D. The t+2D and 2D+t system configurations are visualized in Figure 4.4, while the 2D+t+2D is a combination of both. The spatial wavelet decomposition, referred to by the term '2D', removes spatial redundancy while enabling resolution scalability. The parameter 't' represents the temporal wavelet transform combined with motion estimation, which removes temporal redundancy and enables tem-

poral scalability. Finally, for all system configurations, the wavelet coefficients and motion vectors are entropy coded.

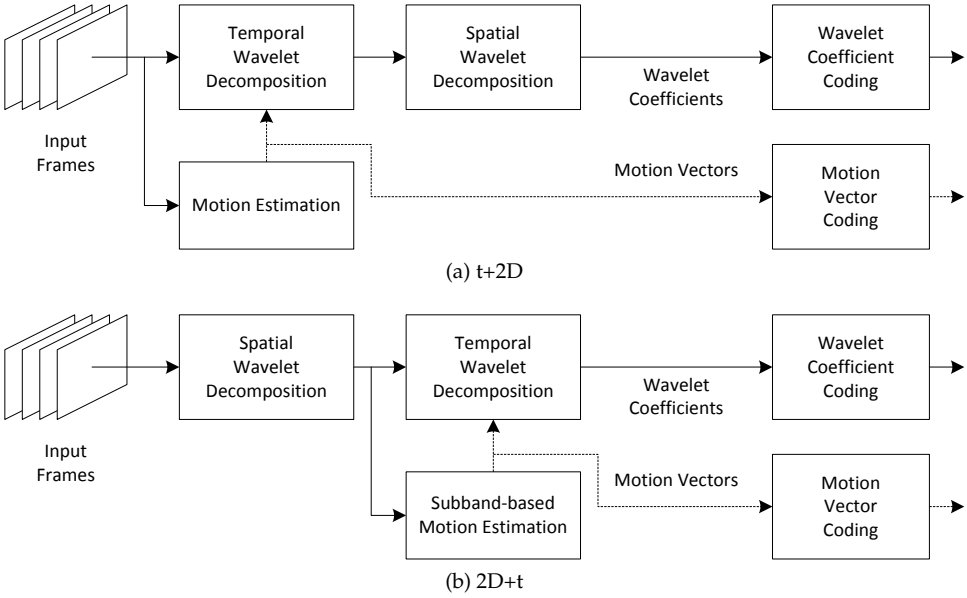


Figure 4.4: The (a) t+2D and (b) 2D+t temporal wavelet coding architectures.

The temporal wavelet transform typically consists of a particular filter type, such as the well-known Haar transform or Daubechies D2 wavelet [21], or the more advanced 5/3 filter [22, 23, 24, 25] schemes, based on motion-compensated lifting. Some schemes even apply different filters at different levels of decomposition and adapt the filter type to the content. For 2D+t and 2D+t+2D schemes, the motion compensation is performed in the wavelet domain and a shift-invariant wavelet transform should be used, such as the complete-to-overcomplete discrete wavelet transform, as proposed by Andreopoulos *et al.* [93, 94, 95, 96].

The temporal wavelet schemes can be also combined with multiple layers, similar to the scalable predictive coder, as discussed in Section 4.2 and visualized in Figure 4.3. For example, a pyramid-based system down-samples the input signal to various resolutions (e.g. 4CIF, CIF and QCIF) and for each of those resolutions, then performs a t+2D decomposition. In this concept, information from low-resolution layers is used as a prediction for high-resolution layers. This layer-to-layer prediction can consist of intra-prediction, motion-vector prediction and residual prediction. Unfortunately, the scalability opportunities inherent to the wavelet transform cannot be exploited, as the layer-to-layer prediction introduces

dependencies between layers. This layer-to-layer dependency forces lower levels to be decoded at full quality, as they are used for prediction of the layers above. This reduces scalability options even further and might even introduce encoder-decoder drift. For these reasons, the combination of layering with wavelet schemes is not considered for our application. The remaining configurations are compared in more detail in the next section.

Selection of suitable coding architecture for surveillance

The previous section has introduced predictive coding, scalable predictive coding based on Laplacian pyramids and three 3D subband coding architectures: t+2D, 2D+t and 2D+t+2D. These coding architectures are compared with complexity and performance trade-offs, which are categorized in Table 4.1, where the symbols indicate the performance for each of the aspects. The table is discussed qualitatively, with a focus on four topics: coding performance, complexity, scalability and features relevant to the domain of video surveillance.

Table 4.1: Performance of various temporal coding architectures for system aspects indicated in the left column. Legend: $--$ = very poor, $-$ = poor, $\bigcirc$ = fair, $+$ = good, $++$ = very good.

Coding / Aspect	Predictive		3D subband		
	Regular	Scalable	t+2D	2D+t	2D+t+2D
Low inherent complexity	$+$	$\bigcirc$	$++$	$-$	$--$
Low motion complexity	$++$	$+$	$++$	$--$	$--$
Coding performance	$++$	$++$	$+$	$+$	$+$
Scalability	$--$	$\bigcirc$	$++$	$++$	$++$
Reduced resolutions	$--$	$++$	$\bigcirc$	$++$	$++$
Low end-to-end delay	$++$	$+$	$+$	$+$	$+$
Trick play and random access	$--$	$-$	$++$	$++$	$++$
Score ($+$ minus $-$)	1	5	10	5	4

Complexity

Both predictive coding systems have a closed loop, which requires a decoder within the encoder, thereby increasing the complexity. The scalable predictive coder is constructed with several layers, in which residual information is propagated through the hierarchical tree. Both predictive coders can utilize straightforward motion es-

timation based on minimum luminance differences, but the scalable version needs to estimate motion at various scales. The 3D subband coders all have an open coding loop, while no decoding step is required during encoding. However, for the 2D+t and 2D+t+2D schemes, the motion compensation is performed in the wavelet domain and a shift-invariant wavelet transform should be used, such as the CODWT (Complete-to-Overcomplete Discrete Wavelet Transform), leading to an increased complexity.

Coding performance
Because of the feedback loop in the predictive coders, quantization errors resulting from the encoding-decoding process are fed back internally. Over time, this allows these coders to correct the errors they have made in the past. Due to this corrective nature, closed-loop coders have a very high coding performance. However, the feedback loop also inevitably impacts scalability, which is discussed next.

Scalability
As the predictive coding system is a closed-loop system, decoding within the encoder should match decoding within the decoder. If scalability is applied to the decoder, the encoder and decoder are not synchronized, so that drift will occur between the encoder and the decoder, leading to severe accumulating artifacts. Bidirectionally predicted frames can be discarded for basic temporal scalability, since they are outside the feedback loop, so that these frames do not lead to accumulating drift artifacts.

Pyramidal-based systems show excellent performance at all resolutions. Because scalability is added through inter-dependent pyramidal levels, only a fixed number of scalability options can be supported for pyramidal systems. This limitation results from the consideration that each additional operation point in the three-dimensional space of quality, spatial resolution and temporal resolution, requires an additional enhancement layer (see Figure 4.3), which significantly increases the coding system complexity. Since the operation space is three-dimensional, the amount of enhancement layers grows rapidly. Scalability is therefore limited to these operational points and their specific order, otherwise encoder-decoder drift will occur.

The 3D subband systems have an open coding loop, so that they do not suffer from encoder-decoder drift. This provides fine-grained scalability, while simultaneously allowing scalability in multiple dimensions, unlike the pyramidal-based systems. Within this class, t+2D systems can show artifacts for reduced spatial resolutions because of non-ideal spatial filters. The non-ideal filters lead to spatial aliasing in the low- and high-frequency subbands. In normal operation, this

aliasing practically becomes zero when wavelet filters are designed to be orthogonal (for example Daubechies 9/7). However, when dismissing high-pass bands for bit-rate or spatial-resolution reduction, the aliasing remains present in the low-pass bands and will cause visible degradations in the image.

Relevant features for video surveillance

End-to-end delay is defined as the time between the capturing moment of the image by the camera until the visualization time at the observer. This delay is in part defined by the coder delay, which can vary, based on the number of 'future' images, which are used to reconstruct a certain frame. Predictive coders generally employ none to two 'future' frames and therefore have a very low end-to-end delay. Because of the feedback loop, prediction errors are fed back and thus reduce over time. The open-loop 3D subband coders do not have this benefit and commonly utilize more frames in a hierarchical structure to reconstruct a single frame. Therefore, the amount of 'future' frames should be carefully considered to limit the amount of future dependencies. These hierarchical dependencies also apply to the scalable predictive coding architecture.

Finally, for surveillance applications, *trick play*, such as reverse and high-speed playback and *random access* are of major concern when reviewing recorded videos. Here, the predictive coders perform rather poorly, due to the temporal dependency resulting from the feedback loop, creating long chains of frame dependencies. Frames in open-loop coders are always depending on a subset of frames, regardless of the playback direction. These dependencies can be biased towards 'past' frames to decrease the end-to-end delay during forward playback, while the argument regarding end-to-end delay is not relevant for reverse playback. The frame dependencies on past frames can be interpreted as a complexity measure for reverse playback.

Selection of the preferred architecture for further system evaluation

Since the focus of this work is on an embedded coding system for surveillance, complexity, scalability and domain-relevant features are of major concern. The numerical score at the bottom of Table 4.1 is accumulated from the number of $+$ and $-$ indications. From this score, we directly conclude that the t+2D architecture is favorable for our application domain. The architecture has low inherent complexity and standard motion estimation and compensation techniques can be applied (see text on complexity). It provides very good scalability in quality, spatial resolution and temporal frame rate (see text on scalability). End-to-end delay can be controlled and trick play and random access are supported (see text on relevant features for video surveillance).

The adopted t+2D architecture does introduce spatial aliasing when decoding at lower resolutions, but this degradation is considered to be acceptable for two reasons. First, in most cases, the aliasing actually provides a visually sharper experience, whereas quality-reducing artifacts only become visible for high-contrast repetitive spatial structures. Second, in our surveillance application, lower resolutions are typically used for overview monitors. In case of important events, decoding and viewing are switched to full resolution, where no spatial aliasing occurs due to the effective use of filters featuring orthogonality.

Therefore, in the remainder of this chapter, the basis for further investigation is the t+2D architecture. The next section will investigate the complexity and performance of various temporal configurations of this architecture.

4.3 Temporal configurations in t+2D coding architectures

This section first provides an overview of the t+2D coding architecture, after which it presents a framework for constructing the structure of various temporal decompositions. For a selection of the best temporal decomposition structure, the section evaluates the dynamic behavior and estimates the computational complexity.

Overview of the t+2D coding architecture

As mentioned in the previous section and visualized in Figure 4.4(a), the t+2D coding architecture consists of a temporal wavelet decomposition with motion compensation, followed by a spatial wavelet decomposition. The temporal wavelet decomposition has the purpose to distinguish static elements in the scene, such as the background, from dynamic elements in the scene, such as a moving car. If this temporal decomposition is performed without motion compensation, the moving car would result in a temporal high-pass representation of the car disappearing in one frame and appearing in the other. The resulting temporal high-pass frame can contain the spatial details of the car in two ways. Depending on the speed of motion, these details would be represented by the difference with the same car at the previous spatial location, or by the difference with the background, which by itself could also contain a large amount of spatial details.

Since many moving objects only have minimal changes of their appearance from frame-to-frame, motion estimation and compensation is commonly utilized, to estimate the motion of moving elements in a block-based fashion. With motion estimation and compensation, only changes to the appearance of moving objects and the revealed details of occluded areas are represented in the temporal high-pass representation. Within the t+2D coding architecture, this motion estimation

and compensation is performed during the temporal filtering process, also known as as Motion Compensated Temporal Filtering (MCTF).

MCTF can be performed with any type of wavelet filter, but simpler types of filters such as the Haar or 5/3 filter are most common. Figure 4.5 shows the motion-compensated temporal filtering process for the low-pass Haar filter. This figure visualizes how motion estimation and compensation are performed, after which the 2-tap high-pass filter is applied to Frame 0 and the motion-compensated Frame 1. For the low-pass filter, processing is identical, but a different filter is employed.

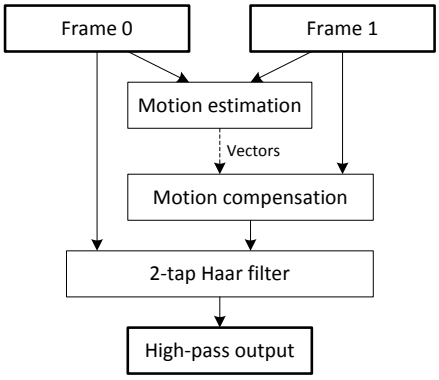


Figure 4.5: Temporal filtering in the t+2D coding architecture using the Haar wavelet.

Similar processing is performed for more complex filters, such as the 5/3 integer wavelet. In a straightforward FIR-filter implementation, the 5/3 temporal wavelet filter has 5 filter taps for the low-pass filter and 3 filter taps for the high-pass filter. The 5/3 temporal filter is commonly implemented using the lifting framework. In this case, the 'predict' step results in the high-pass output, utilizing a 3-tap filter (identical to the non-lifting 3-tap filter). The successive 'update' step results in the low-pass output, utilizing a different 3-tap filter. This 5/3 temporal filtering structure with lifting is visualized in Figure 4.6(a). Similar to the Haar filter, motion estimation and compensation is utilized prior to applying the 3-tap filter. This notation of the temporal structure becomes inconvenient when representing larger structures and multiple temporal decompositions. Therefore, in the following, larger structures are denoted and visualized in a shorthand representation, as depicted in Figure 4.6(b). In this shorthand notation, the complete motion estimation and compensation structure is represented by the diagonal arrow.

The basic 5/3 filter structure can be expanded to multiple temporal decompositions, by successively filtering the temporal low-pass output. For multiple decompositions, this structure becomes rather complicated. For example, for the 5/3

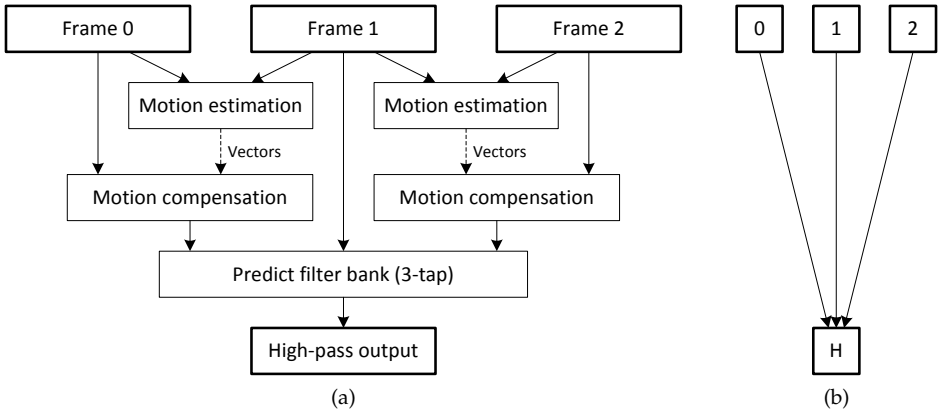


Figure 4.6: Temporal filtering in the t+2D coding architecture using the 5/3 wavelet filter. (a) Full structure and (b) shorthand notation with motion estimation/compensation structure represented by the diagonal arrows, with filtering implicitly part of the processing.

wavelet with only 2 temporal decompositions, the low-pass frames depend on 17 input frames. This aspect clearly violates our requirements of low complexity and low end-to-end delay. Therefore, in the next section, a framework for constructing decomposition structures with varying temporal depth is explored, which also allows various structures within the same temporal depth.

Framework for varying temporal decomposition structures

Within the t+2D architecture class, a complexity-performance trade-off can be made by varying the temporal decomposition structure. Temporal decomposition is performed using Motion Compensated Temporal Filtering (MCTF), where motion compensation is integrated within the lifting steps of the temporal wavelet filter. Different temporal configurations can be explored, depending on which 'predict' and 'update' steps are included. Among the various proposals of MCTF, the Unconstrained Motion Compensated Temporal Filtering (UMCTF) by Van der Schaar and Turaga [67] is adopted for the following reasons. First, the structure utilizes three simple building blocks that can be flexibly used to design a wide variety of temporal configurations. Furthermore, by selectively limiting predict and update steps from future frames, frame dependencies are reduced, so that encoder, decoder and end-to-end delay can be controlled.

In addition to the three building blocks proposed in UMCTF, we propose a fourth basic building block. The four building blocks now available for constructing various temporal configurations are visualized in Figure 4.7.

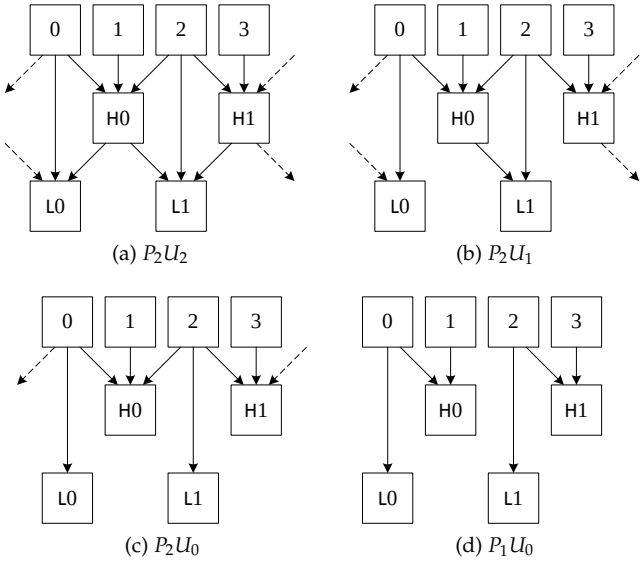


Figure 4.7: Four methods of implementation of predict and update steps: (a) double predict, double update (P_2U_2); (b) double predict, single update (P_2U_1); (c) double predict, no update (P_2U_0) and (d) single predict, no update (P_1U_0).

Let us use the following notation, as defined by Pau *et al.* [71]. Parameter x_t denotes the input video frame at time t, parameter l_t stands for the temporal low-pass subband frame and h_t represents the temporal high-pass subband frame. As motion-compensated lifting is used, predictions of odd frames x_{2t+1} are made from even frames x_{2t} and x_{2t+2}. Two motion vector fields are estimated: (1) forward predicted from x_{2t} to x_{2t+1}, denoted by $\mathbf{v}^{+}_{2t+1}$ and (2) backward predicted from x_{2t+2} to x_{2t+1}, denoted by $\mathbf{v}^{-}_{2t+1}$. Let $\mathbf{n}$ be the 2D spatial position in a frame in vectorized form and x the corresponding pixel at position $\mathbf{n}$, then the motion-compensation operator $\mathcal{C}$ is defined by $\mathcal{C}(x, \mathbf{v}_t)(\mathbf{n}) = x(\mathbf{n} - \mathbf{v}_t(\mathbf{n}))$. The update step requires inversion of the motion-compensation operator, which is represented by $\mathcal{C}^{-1}$. Due to the non-invertibility of $\mathcal{C}$, the operator $\mathcal{C}^{-1}$ is not defined everywhere and special care needs to be taken for unconnected pixels as discussed in [90] and [97]. It is also possible to circumvent the problem with the $\mathcal{C}^{-1}$ operator, by omitting the update steps from the lifting altogether. The merits of this will become clear in the upcoming discussion.

Furthermore, let us denote the predict and update steps by the terms P and U, respectively. With this notation, a predict step P that is used i times for constituting a high-pass stage is indicated by P_i, while U_j constitutes a low-pass stage with j

update steps. The four different temporal filtering methods P_2U_2, P_2U_1, P_2U_0 and P_1U_0 are shown in Figure 4.7. These temporal filtering methods (in the same order) are presented in a formal expression by Equations (4.1) through (4.4), giving

$$\begin{aligned} h_t &= x_{2t+1} - \frac{1}{2}\big(\mathcal{C}(x_{2t}, \mathbf{v}^+_{2t+1}) + \mathcal{C}(x_{2t+2}, \mathbf{v}^-_{2t+1})\big), \\ l_t &= x_{2t} + \frac{1}{4}\big(\mathcal{C}^{-1}(h_{t-1}, \mathbf{v}^-_{2t-1}) + \mathcal{C}^{-1}(h_t, \mathbf{v}^+_{2t+1})\big), \end{aligned} \tag{4.1}$$

$$\begin{aligned} h_t &= x_{2t+1} - \frac{1}{2}\big(\mathcal{C}(x_{2t}, \mathbf{v}^+_{2t+1}) + \mathcal{C}(x_{2t+2}, \mathbf{v}^-_{2t+1})\big), \\ l_t &= x_{2t} + \frac{1}{2}\mathcal{C}^{-1}(h_{t-1}, \mathbf{v}^-_{2t-1}), \end{aligned} \tag{4.2}$$

$$\begin{aligned} h_t &= x_{2t+1} - \frac{1}{2}\big(\mathcal{C}(x_{2t}, \mathbf{v}^+_{2t+1}) + \mathcal{C}(x_{2t+2}, \mathbf{v}^-_{2t+1})\big), \\ l_t &= x_{2t}, \end{aligned} \tag{4.3}$$

$$\begin{aligned} h_t &= x_{2t+1} - \mathcal{C}(x_{2t}, \mathbf{v}^+_{2t+1}), \\ l_t &= x_{2t}. \end{aligned} \tag{4.4}$$

It is now possible to construct an N-level temporal transform with the four configurations of Figure 4.7. Using the three configurations originally proposed by Pau *et al.* [71], P_2U_2, P_2U_1 and P_1U_0, the transform can be parameterized by (P, Q), with P denoting the number of P_2U_2 transforms at the finest level(s) and Q denoting the number of P_2U_1 transforms at the following level(s). If any, the remaining level(s) use the P_1U_0 transform. When employing the fourth configuration P_2U_0, additional alternative temporal transform configurations can be constructed, which are discussed in the next section.

Alternative temporal transform configurations

Besides the above configuration, additionally two alternative transform configurations are proposed, which omit all update steps and both only consist of the P_1U_0 and P_2U_0 building blocks. The first alternative configuration consists of bidirectional Prediction Only and is called PO. The second alternative configuration employs bidirectional prediction for the two lower levels, while single forward prediction is utilized at the remaining higher levels. This configuration has a significantly lower end-to-end delay and is therefore called BDLD, which stands for

BiDirectional Low Delay. This second alternative is depicted in Figure 4.8 for four temporal levels, where input frames are depicted by bands 0-15 and output frames by bands: LLLL, H0, LH0, H1, LLH0, H2, LH1, H3, LLLH, H4, LH2, H5, LLH1, H6, LH3 and H7.

The structure of the BDLD configuration as seen in Figure 4.8 is based on several design decisions. At lower decomposition levels, frame differences due to motion are smaller than for higher levels. This simplifies motion estimation and improves its accuracy for a broad range of possible algorithms. As a result, at these lower levels, it is possible to achieve good temporal decorrelation using bidirectional prediction. Furthermore, because the time difference between frames is still relatively small, the introduced frame dependencies only marginally contribute to the end-to-end delay. For the higher levels, motion estimation becomes more difficult due to the increased temporal distance. Propagation of frame dependencies is significant, where forward frame-dependencies can lead to significant time delays. Therefore, at these higher levels, only uni-directional prediction from past frames is utilized. This limits frame dependencies and prevents further growth of the end-to-end delay (without forward dependencies, it is not needed to wait for a frame to become available). In Figure 4.8, it can be observed that the H0 band can only be calculated after input Frame 4 becomes available. This results in an encoding delay of two frames, due to the dependency of the LH0 band on the input two frames ahead. This critical dependency is visualized with bold arrows.

Finally, no update steps are applied in both the PO and the BDLD configuration. The reasons for omitting the update steps are multifold. First, the update steps alter the low-pass frames, by updating the input frames using high-pass frames. In theory, this could function as a form of temporal noise reduction, which would slightly improve the coding performance. In practice, imperfect motion estimation introduces artifacts in the low-pass frames, which reduce the visual quality and increase the bit rate. Furthermore, in a situation where the high-pass frames are discarded (such as low frame-rate playback and storage-space reduction), these artifacts remain in the output. Another argument is that the update steps introduce additional dependencies between frames, increasing the encoder delay and end-to-end delay. Finally, without update steps, an additional benefit is no longer requiring the complex process of inverting the motion-compensation operator.

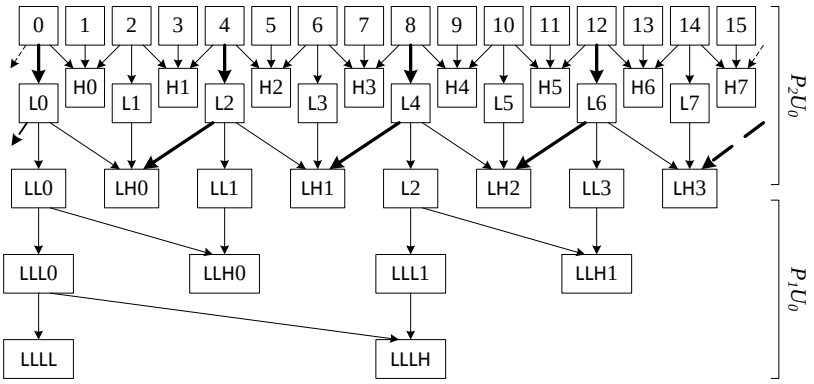


Figure 4.8: The four-level BiDirectional Low Delay (BDLD) temporal configuration. Bold arrows indicate the critical dependency paths for encoder delay.

Analysis of the dynamic behavior of the proposed temporal configurations

To evaluate the system complexity of the proposed framework to control the temporal configurations of the 3D subband coder, various system implementation aspects of the coder are analyzed for different temporal configurations. Several performance metrics (e.g. motion-estimation computations, dynamic memory usage, etc.) are calculated by simulating the dynamic behavior of the subband computation process of the encoder and the decoder for a Group Of Pictures (GOP) of 16 frames. This dynamic behavior is visualized for the BDLD temporal configuration of the encoder in Figure 4.9 and of the decoder in Figure 4.10. The dynamic behavior is shown for all 46 subbands, whereby each band consists of three columns: write, memory and read. The rows depict the entering time at which bands are written to memory (red block in left column), the time interval residing in the memory (blue block in center column), or the reading time from memory (green block in right column). The dynamic behavior of the BDLD configuration takes 19 time slots for completion, which means the last three time slots overlap with the first three from the next GOP of 16 frames.

From $t = 1$ to $t = 16$, it can be seen that the input frames (Bands 0-15) are written to the frame memory for further calculations. One of these calculations is the construction of Band H0 from Bands 0, 1 and 2. In the top sub-figure of Figure 4.9, reading of the input data can be observed at $t = 3$, where Bands 0, 1 and 2 are read for motion estimation and compensation. The resulting Band H0 is constructed and written to memory, as seen in the middle sub-figure of Figure 4.9.

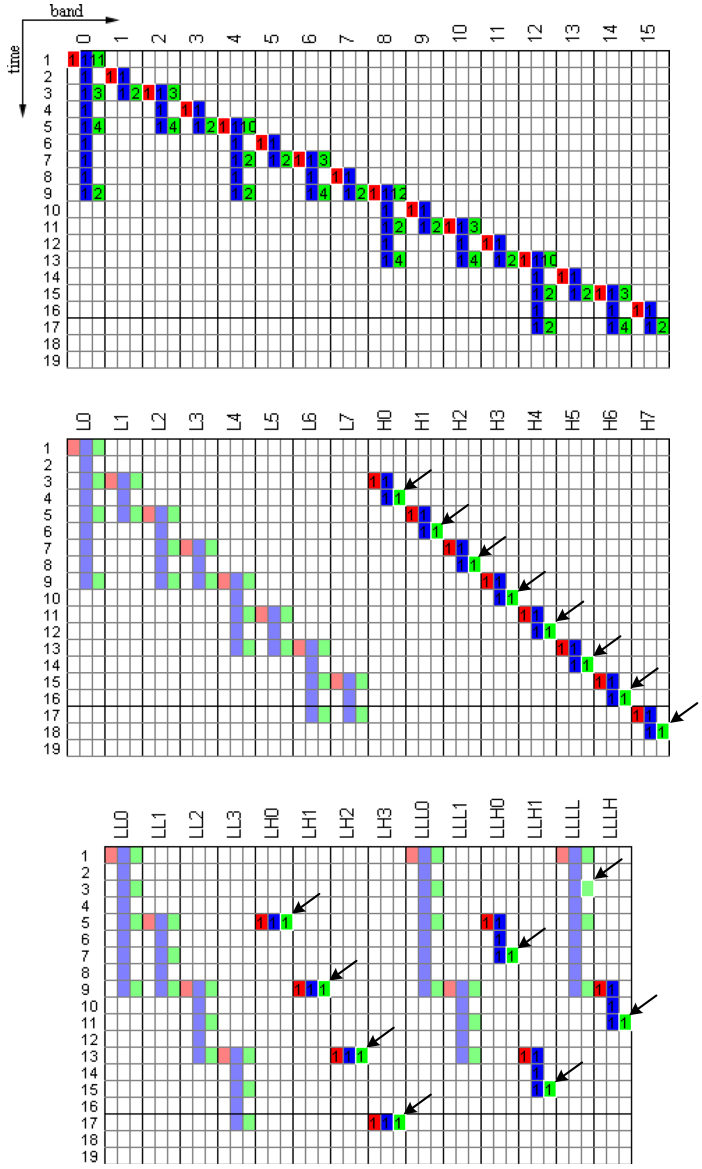


Figure 4.9: Dynamic behavior of the encoder for the BDLD temporal configuration. Red blocks indicate a write action to memory, green blocks a read action from memory and blue blocks indicate the storage duration of a band in memory. Numbers inside the block indicate the number of read/write occurrences. Blocks with low saturation and without numbers indicate that a duplicate band exists, so that the current band does not need to be stored in memory. Arrows to green blocks indicate read actions for transmitting the band.

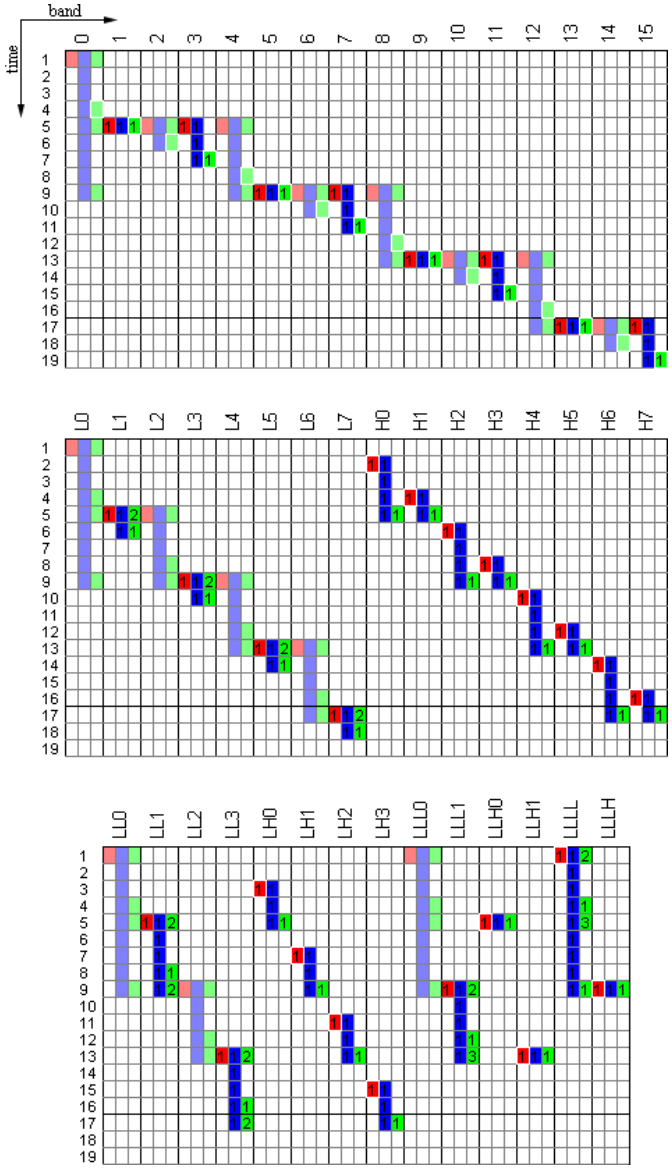


Figure 4.10: Dynamic behavior of the decoder for the BDLD temporal configuration. Red blocks indicate a write action to memory, green blocks a read action from memory and blue blocks indicate the storage duration of a band in memory. Numbers inside the block indicate the number of read/write occurrences. Blocks with low saturation and without numbers indicate that a duplicate band exists, so that the current band does not need to be stored in memory.

At $t = 3$, also the first output is generated by sending Band LLLL, as indicated by the arrow to the green block at the bottom of Figure 4.9.

Since the BDLD temporal configuration does not use update steps, low-pass bands are identical to their center input. For example, Band L0 is identical to Band 0, LL0 to L0, LLL0 to LL0, etc. When bands are identical, only one instantiation needs to be stored, in this example, Band 0. The identical bands (L0, LL0, LLL0 and LLLL) are not stored in memory and shown as opaque copies of Band 0 in Figure 4.9.

Bands LH0-LH3 at the bottom of Figure 4.9 are the start of the critical delay path of the encoder, as indicated by the bold arrows in Figure 4.8. The bottleneck of the path can be directly noticed from Figure 4.9, because after the calculation and storage of Bands LH0-LH3, they are immediately read by the entropy coder and transmitted. This is shown by the small cluster of boxes for Bands LH0-LH3.

These memory and bandwidth behavior diagrams have been explored for 7 different coding configurations. It is readily understood that these diagrams provide important system parameters such as the maximum memory usage, the number of memory accesses, etc. The following section discusses the results of comparing these different coding configurations and the related implementation aspects.

Analysis of computational complexity

The results of the analysis of the dynamic behavior of the proposed temporal configurations are summarized in Table 4.2. For the remaining configurations, a dynamic analysis has been performed similar to the one discussed in the previous section and visualized in Figures 4.9 and 4.10 for the BDLD temporal configuration. Due to their size, the diagrams of these configurations are omitted in this chapter, but their resulting numerical performance metrics are given in Table 4.2 and Table 4.3. The calculation of the number of frames required for arbitrary frame access is worked out in Appendix D.

In this table, the most important performance metrics on computation and memory usage are listed by denoting the number of processed frames per GOP for a computing task (top of column) and a specific temporal configuration (see row at the left). For example, the (4,0) temporal configuration performs motion estimation on 30 frames and motion compensation on 60 frames. This process produces 30 motion vector fields and requires a maximum of 41 frames concurrently stored in memory. Per GOP, 46 intermediate frames are stored to memory and these frames are accessed 106 times in total. Finally, to reconstruct a single random output frame, on the average, 13 entropy-coded frames need to be decoded.

Table 4.2: Quantified performance metrics for different temporal configurations. Numerical results indicate the involved number of frames per GOP required for the computing task given at the top of the considered column. The most right column refers to the average number of retrieved encoded frames for an arbitrary output frame.

Temporal Configuration	Motion Est.	Motion Comp.	Vector Fields	Mem. Usage	Write Access	Read Access	Average Access
(4,0)	30	60	30	41	46	106	13
(1,2)	29	51	29	17	45	96	8.5
(0,2)	27	39	27	11	43	86	6
(0,1)	23	31	23	8	39	69	4.5
(0,0)	15	15	15	4	31	46	3
PO	30	30	30	32	46	76	5
BDLD	27	27	27	8	43	70	3.8

The numbers regarding motion and memory usage are easily understood, but a closer inspection on the average access is desirable. Arbitrary frame access is the most critical feature in resource usage when searching in a recording database. For example, an average access of 13 frames means that on the average, it is required to fetch 13 encoded frames and perform the complete temporal decomposition, to reconstruct a single output image. This requirement favors the choice for the (0,0) and BDLD configurations.

Another important aspect in video surveillance is the end-to-end delay of the video encoder-decoder chain, especially for the active control of PTZ cameras. Table 4.3 provides the encoder, decoder and cascaded delays in number of frames for different temporal configurations. The (4,0) configuration has the largest delay of 45 frames which, at 30 fps, equals to an end-to-end delay of 1.5 seconds. The (0,2), (0,1), (0,0) and BDLD configurations stay under 3 frames or equivalently 150 milliseconds, which are suitable for surveillance applications.

When comparing the temporal configurations, both tables show that the (4,0) configuration is clearly more expensive than the other proposals. This is because of the large memory requirements and the high number of frames required to decode a single random frame (See Table 4.2), combined with the large end-to-end delay (See Table 4.3). Therefore, we do not consider it a candidate for our application, aiming at low complexity, scalability and domain-specific features. For the other configurations, i.e. the (0,2), (0,1), (0,0) and BDLD configurations, more feasible system parameters are obtained, such as a significantly lower delay and memory requirements. Hence, a further comparison in quality is required which is performed later in Section 4.5.

Table 4.3: Processing delays counted in frames for different temporal configurations.

Temporal Configuration	Encoder Delay	Decoder Delay	Cascaded Delay
(4,0)	30	23	45
(1,2)	6	7	9
(0,2)	2	3	3
(0,1)	1	1	1
(0,0)	0	0	0
PO	8	15	15
BDLD	2	3	3

During the investigation of the complexity and the upcoming performance analysis of the remaining alternatives in Section 4.5, a few shortcomings in the t+2D coding framework were found, which are addressed in the following section including extensions for improvement.

4.4 Extensions to the temporal coding structure

Shortcomings in the t+2D coding framework

The analysis of the t+2D coding framework has revealed two shortcomings. The first involves energy correction of the temporal wavelet transform. As observed for images in Chapter 3, the output of the wavelet transform needs proper energy balancing between low- and high-pass components for good rate-distortion performance. The same holds true for wavelet filtering in the temporal domain (explained later). Another important aspect is that the energy correction in the temporal hierarchical tree can lead to rounding error propagation within the tree when fixed-point calculations are used. The implementation of the energy correction requires precise scaling operations with floating-point precision.

The second shortcoming of t+2D coding structures is a strong fluctuation of the quality of frames within a GOP. Within a GOP, specific frames are referenced multiple times in the temporal hierarchical tree, such as the first frame of the GOP. Because of this multiple referencing, these frames are therefore encoded at a higher quality. This seems reasonable because many frames are directly and indirectly derived from these reference frames. However, in an open-loop scalable coding framework, the decoder will also decode these reference frames at a lower quality, so that a high-quality reference cannot be reconstructed.

Therefore, the following sections concentrate on improving the above two shortcomings and discuss two extensions to the temporal coding structure, which deviates from the standard t+2D structure. The first extension, Temporal Energy Correction (TEC), isolates the temporal energy correction in the leafs of the temporal decomposition tree and reduces computational complexity. The second extension, Low-Complexity Encoder Feedback (LCEF), introduces a low-complexity quality reduction for the first frame of the GOP, which results in an improved average quality and reduced quality fluctuations. When LCEF is applied in the encoder, the coded stream is compatible to the normal case so that decoder modifications are not required.

Temporal Energy Correction (TEC)

To improve the energy balance of the temporal transform, each iteration in the temporal lifting process can perform an energy correction as part of the lifting process, as discussed in Chapter 2. Due to the successive use of lifting in the temporal configuration, energy correction is performed at multiple points in the tree (in the nodes), as indicated in Figure 4.11(a) for the BDLD configuration. The energy correcting factors α for the low-pass and β for the high-pass energy correction are usually floating-point numbers, while many highly optimized implementations use fixed-point arithmetic. If implemented in a straightforward way, several conversions between fixed- and floating-point are necessary, thereby introducing cumulative conversion errors.

It is therefore proposed to concentrate all Temporal Energy Correction (TEC) to the point just prior to the start of entropy coding (in the leafs). As a result, uncorrected (and thus original) input frames are used for the predict steps at the leafs of the temporal configuration and only the final output is energy corrected. This concentration of energy scaling at the last stage is only applicable to temporal configurations without update steps, such as the BDLD configuration, because the intermediate frames are not altered through update steps.

The derived correction factors are visualized in Figure 4.11(b). A unity factor represents the absence of any computation and is included in the figure solely for explanatory purposes. It can be easily observed from the figure, that the total scaling/energy correction of the filtered frames does not change. For example in Figure 4.11(a), when following the path from Frame 0 in a downwards direction, the factor α is four times applied, which is equivalent to the single scaling of factor α^4 in Figure 4.11(b) for the same path.

Furthermore, as a result from applying TEC, the memory consumption of the complete coder is reduced significantly, which occurs for two reasons. First, inter-

mediate low-pass results are not calculated and therefore do not need to be stored. Second, the input frames for lifting are all original frames in their original fixed-point precision (e.g. 8 bits), enabling minimal memory size and bandwidth. All modifications made to the encoder, can also be applied to the decoder and the discussed complexity benefits apply to the decoder as well.

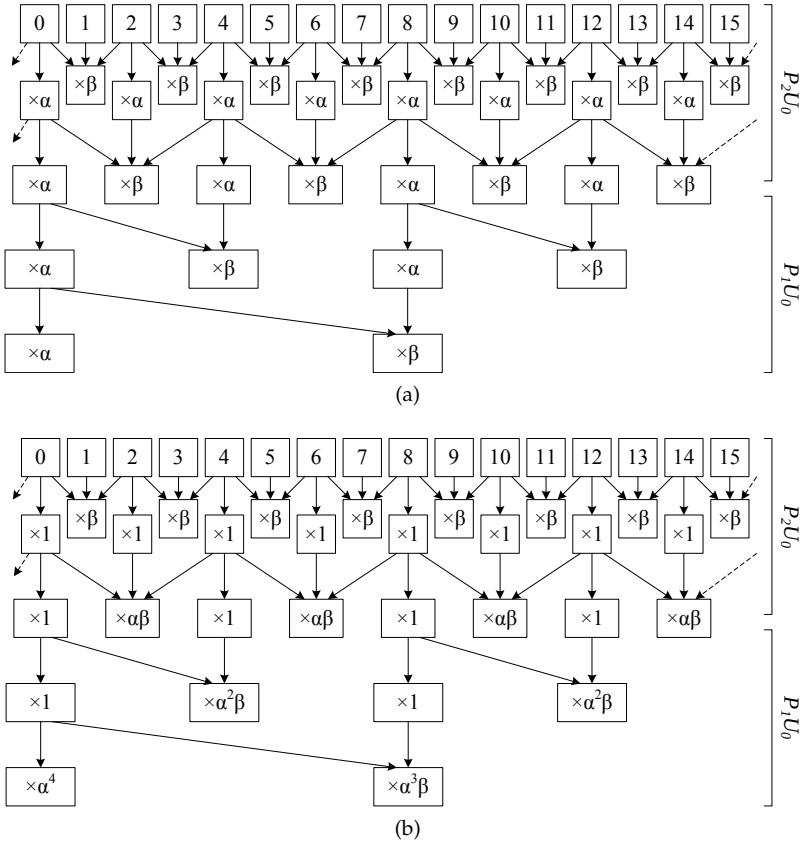


Figure 4.11: Temporal Energy Correction (TEC) applied to the BDLD temporal configuration in the (a) nodes and (b) leafs of the hierarchical tree.

Low-Complexity Encoder Feedback (LCEF)

Visual quality of scalable video codecs can fluctuate over time, due to the periodic GOP structure. The intraframe-coded picture is coded at a higher quality, since all frames are derived from this picture within a GOP. To achieve a more constant quality over time, a novel extension is proposed, called Low-Complexity Encoder

Feedback (LCEF). Besides offering a more constant visual quality, it also improves the average quality. This is obtained without adaptations of the decoding algorithm, hence only the encoder is modified.

The novel encoder modification consists of the addition of a feedback loop for the first frame of the GOP only. In this feedback loop, the first frame of the GOP is reduced in quality, in a manner that is equivalent to the quality-reduction operation in the decoder. This Quality-Reduced Frame (QRF) is a more accurate representation of the reference used in the decoder when performing lossy decoding.

This quality reduction in the encoder limits the highest quality that can be decoded and LCEF is therefore only applicable to lossy coding systems. However, due to storage and bandwidth constraints in surveillance systems, lossy coding is the only practical solution.

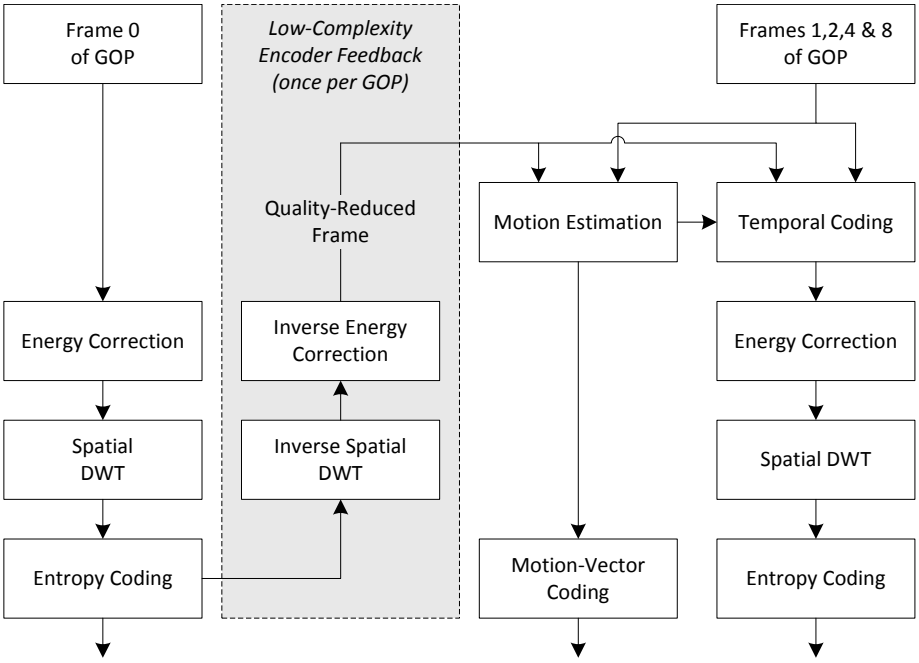


Figure 4.12: Part of the SVC which is modified by the Low-Complexity Encoder Feedback (LCEF) to generate a Quality-Reduced Frame (QRF) within the encoding process. The complexity added by LCEF is indicated by the grey box.

Figure 4.12 shows how LCEF is embedded within the temporal encoding framework. The feedback loop is only applied to the first frame in each GOP to minimize complexity. The first frame of the GOP is always directly encoded, without

any temporal filtering, due to the omission of update steps in the BDLD temporal configuration. In this special case, the quality reduction of the first frame can be applied without performing a complete entropy encoding and decoding step. Instead, the wavelet coefficients are quantized in the same way as the scalable entropy encoder, i.e. by bit-plane truncation, followed by the inverse Discrete Wavelet Transform (DWT) to obtain the QRF. The QRF will be used as a reference for the temporal coding and, if desired, for the motion estimation. Effectively, the only complexity added is a single inverse DWT and the inverse energy correction indicated in Figure 4.12 by the grey box. This is because the forward DWT and wavelet coefficient quantization are performed as part of the regular encoding framework.

An evaluation test has been conducted to distinguish whether motion estimation should be performed from the QRF instead of the original frame. It has been found that the performance is slightly decreased when using the QRF for motion estimation, probably due to the asymmetry of the motion estimation with a QRF frame at one side and an original frame at the other side. Therefore, in LCEF, motion estimation is performed using the original frames.

4.5 Experimental results

This section evaluates the performance of the proposed temporal framework using several standard video sequences. These test sequences were selected for the following reasons. All sequences are available in raw YUV format in the public domain. The chosen sequences also approximate typical video surveillance applications, such as tracking an object of interest. Furthermore, a variety of common camera resolutions have been chosen to represent variations and developments in video surveillance, from standard-definition 4CIF (704×576 pixels, 1:1[2]) to high-definition videos at 720p (1280×720 pixels, 1:1) and 1080p (1920×1080 pixels, 1:1). An overview of the utilized sequences is given in Table 4.4 and in Appendix E, where detailed descriptions for each of the sequences can be found. Initially, results are presented for all sequences, but in later discussions this is reduced to only the most common subset of sequences (City and Crew). These two sequences are well known and experiments have shown that they adequately illustrate the items under experimentation.

The experimental results are grouped by topic. First experiments highlight the correlation between quality and temporal configurations, followed by the merits of including update steps in the temporal tree. The third experiment investigates

[2]The progressive format is abbreviated to video shorthand notation 1:1.

Table 4.4: Details of the raw videos used for experimentation. All videos are progressive.

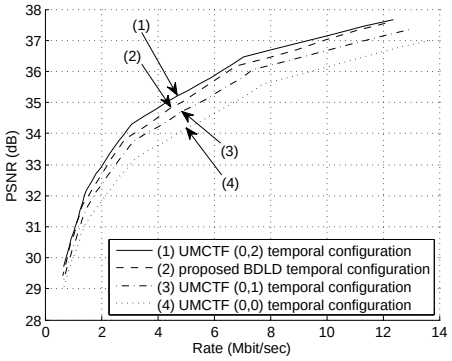
Video sequence	Color space / sub-sampling	Resolution [pixels]	Aspect Ratio	Frame rate [fps]	Length [frames]
City	YC_bC_r / 4:2:0	704×576	4:3	60	600
Crew	YC_bC_r / 4:2:0	704×576	4:3	60	600
Parkrun	YC_bC_r / 4:2:0	1280×720	16:9	50	504
Mobcal	YC_bC_r / 4:2:0	1280×720	16:9	50	504
Stockholm	YC_bC_r / 4:2:0	1280×720	16:9	59.94	604
Station 2	YC_bC_r / 4:2:0	1920×1080	16:9	25	313
Crowd run	YC_bC_r / 4:2:0	1920×1080	16:9	50	500

motion estimation and vector coding, after which the quality is examined when decoding the bitstream at a reduced resolution. In a fifth test, the performance of TEC (Temporal Energy Correction) and LCEF (Low-Complexity Encoder Feedback) are evaluated. This section concludes with a system comparison of the full modified system against a state-of-the-art H.264 SVC codec.

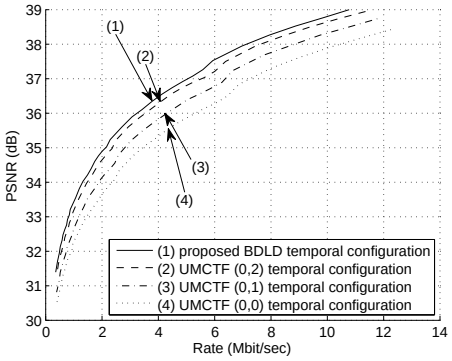
Quality comparison of different temporal configurations

Figure 4.13 shows the rate-distortion curves of the various temporal configurations with low end-to-end delay properties ($t_{delay} \leqq 3$ frames) for several test sequences. For now, the results in this figure omit the cost of motion-vector coding to highlight the resulting quality differences between the different temporal configurations. From these curves, it can be concluded that a more complex temporal configuration also entails a better coding performance, which can be observed by the higher PSNR at the same bit rate.

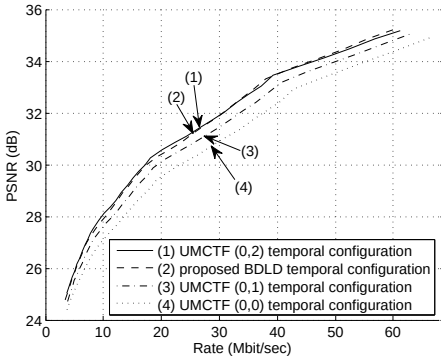
When incorporating the cost of the motion vectors, the rate-distortion curves of Figure 4.14 are obtained. The immediate consequence is that all curves shift somewhat to the right (higher bit rate), giving a slightly reduced PSNR for the same bit rate. However, temporal configurations with a low count of motion vectors, such as (0,0), are penalized less by the extra bit rate of the motion vectors. Especially at low bit rates (below 2 Mbit/s for 4CIF, 60Hz), the cost of lossless motion-vector coding becomes a bottleneck. Since the (0,0) configuration exploits only 15 motion vectors per GOP, versus 27-30 vectors for the other configurations, it provides the highest performance for low bit rates. When low bit rates are required, it could be appropriate to adopt a lossy motion-vector coding scheme, so that an optimal division (in the rate-distortion sense) of available bandwidth between coefficient



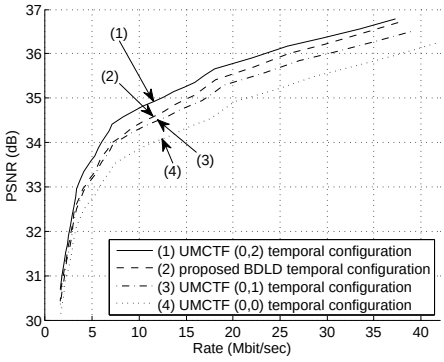
(a) City 704 × 576 pixels, 60 fps



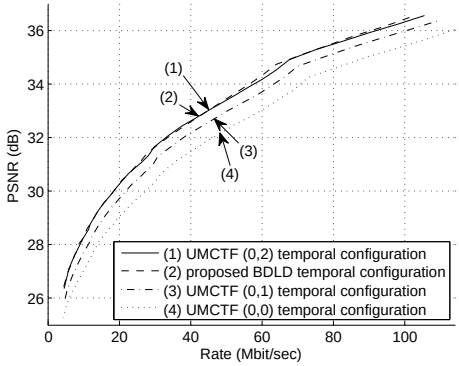
(b) Crew 704 × 576 pixels, 60 fps



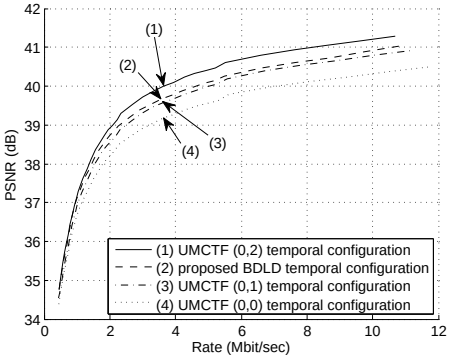
(c) Park run 1280 × 720 pixels, 50 fps



(d) Stockholm 1280 × 720 pixels, 60 fps

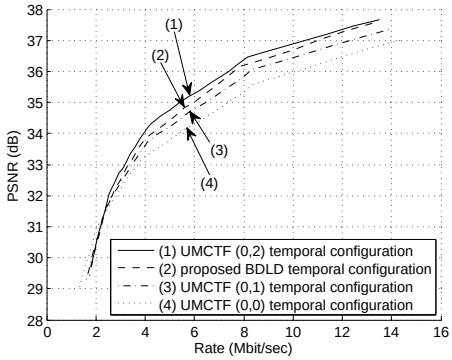


(e) Crowd Run 1920 × 1088 pixels, 50 fps

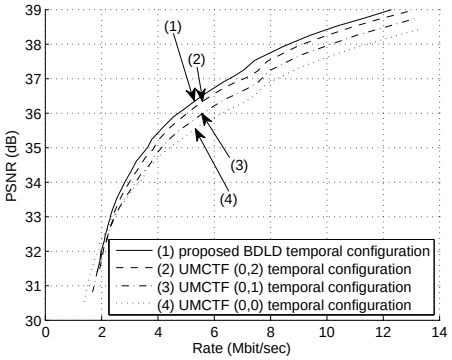


(f) Station 2 1920 × 1088 pixels, 25 fps

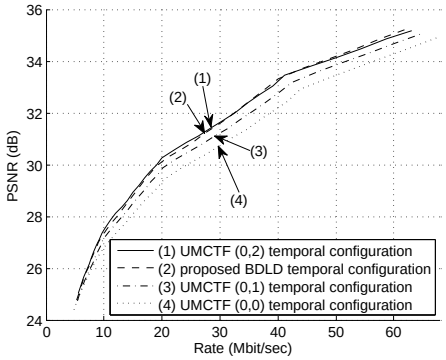
Figure 4.13: Comparison between the (0,0), (0,1), (0,2) and BDLD temporal configurations for the (a) City, (b) Crew, (c) Park Run, (d) Stockholm, (e) Crowd Run and (f) Station 2 sequences, without motion-vector coding.



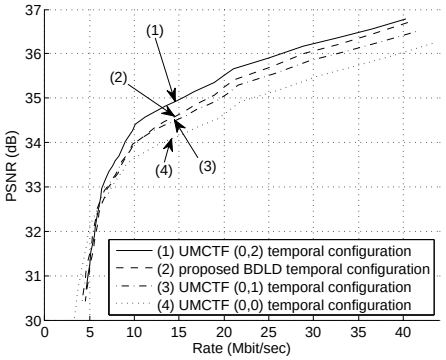
(a) City 704 × 576 pixels, 60 fps



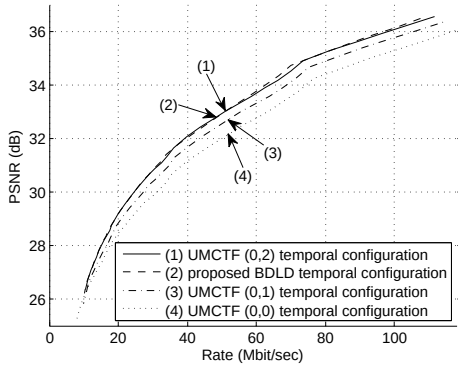
(b) Crew 704 × 576 pixels, 60 fps



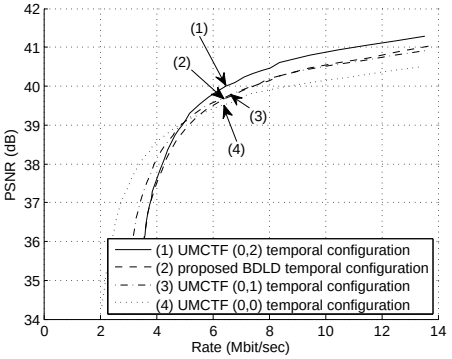
(c) Park run 1280 × 720 pixels, 50 fps



(d) Stockholm 1280 × 720 pixels, 60 fps



(e) Crowd Run 1920 × 1088 pixels, 50 fps



(f) Station 2 1920 × 1088 pixels, 25 fps

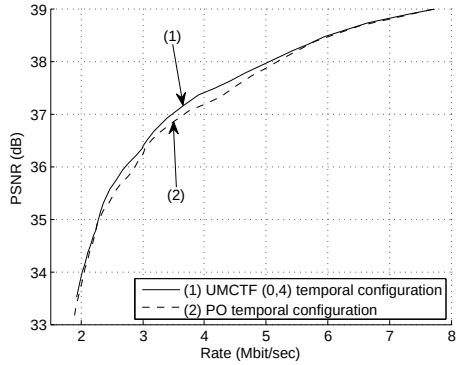
Figure 4.14: Comparison between the (0,0), (0,1), (0,2) and BDLD temporal configurations for the (a) City, (b) Crew, (c) Park Run, (d) Stockholm, (e) Crowd Run and (f) Station 2 sequences, with motion-vector coding.

coding and motion-vector coding can be made. For example, one could propose to encode the motion vectors using a scalable coder, similar to the wavelet coefficients. Unfortunately, this is not possible because severe encoder-decoder drift will occur. In the encoder, the original motion vectors are utilized inside the temporal lifting steps to create a motion-compensated frame, which is used to create a temporal high-pass frame. If quantized motion vectors are used in the decoder, a severe mismatch for the temporal filtering will occur, especially around object boundaries. Moreover, the previous ideas not only require the choice of a best operating bit rate, but could also lead to significant distortions. Both of which are undesirable in scalable video coding.

A better conceptual solution for rate-distortion optimization including motion-vector processing, would be to utilize multiple layers of motion information and frame-difference information, but as discussed earlier in Section 4.2, for complexity reasons these multi-layer solutions are not used in our SVC.

Merits of including update steps

From the rate-distortion plots of Figure 4.13, it can be observed that update steps in UMCTF improve the PSNR performance. Even though the PSNR increases by using update steps, they can still cause severe visual artifacts. An example of such artifacts is shown in Figure 4.15(d) (see the block noise on the grey wall in the middle of the picture). From the rate-distortion curve in Figure 4.15(a), it can be noticed that the curve for the (4,0) temporal configuration (UMCTF) has a higher PSNR than the PO configuration. These two configurations have identical predict steps, but (4,0) utilizes all update steps, while PO has no update steps. When looking at the RD curve at 3 Mbit/s, the PSNR of the (4,0) system is 0.22 dB higher than that of the PO configuration. A visual examination is provided by a part of Frame 3 of the Crew sequence in Figure 4.15(b). The same frame encoded with both the (4,0) and the PO configuration, is presented in Figure 4.15(d) and Figure 4.15(c), respectively. In contrast with the previous PSNR discussion, the visual quality of the PO configuration outperforms the (4,0) configuration. The (4,0) configuration shows severe artifacts caused by the update steps, which originate from the photography flashes in another frame of the sequence, which propagate to the indicated frame through the update steps. In a surveillance application, the LLLL band (significantly affected by update steps) is used for long-term storage, while other temporal subbands can be dismissed over time. Therefore, we conclude that distortions in the important temporal low-pass frames are not acceptable, so that we block the use of update steps in our video coder to prevent such distortions.



(a)

(b)

(c)

(d)

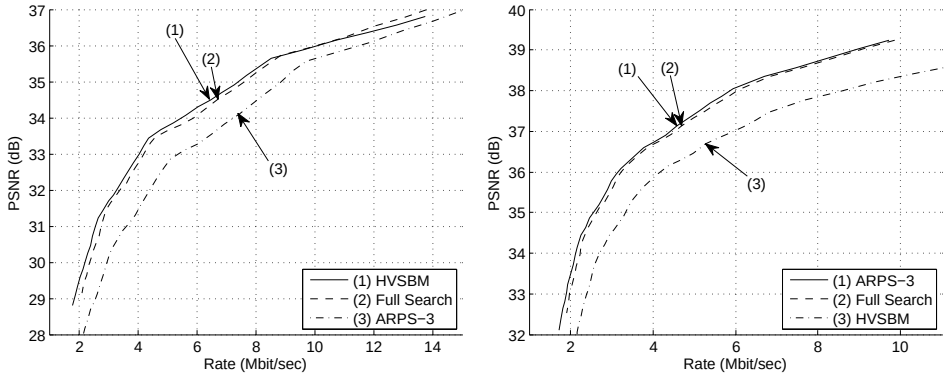
Figure 4.15: Comparison of PO and UMCTF (4,0) temporal configurations: (a) rate-distortion curves, top-left corner of Frame 3 Crew sequence (b) original, (c) encoded using PO @ 3 Mbit/s and (d) encoded using UMCTF (4,0) @ 3 Mbit/s. Amplitudes have been magnified by a factor of two to make the artifacts more visible and independent of printing.

Motion estimators and vector coding

To investigate the complexity and performance of different types of motion estimation and vector coding, three different Motion Estimators (ME) and two types of coding are implemented: (1) fixed block size (16×16) full-search ME with RVLC motion-vector coding [98], (2) fixed block size (16×16) ARPS-3 fast-search ME [99] with RVLC motion-vector coding and (3) variable size ($4 \times 4 - 64 \times 64$) HVSBM hierarchical ME with Golomb-Rice motion-vector coding [64].

The employed full- and fast-search algorithms are broadly accepted, whereas the hierarchical motion estimation is more state-of-the-art and fits well with the

scalable wavelet coder because the motion blocks are split using quadtrees. Figure 4.16 shows the rate-distortion curves for City and Crew sequences encoded with the FS (Full-Search), ARPS-3 (Unequal-arm Adaptive Rood Pattern Search) and HVSBM (Hierarchical Variable Size Block Matching) motion estimators.



(a) City 704 × 576 pixels, 60 fps

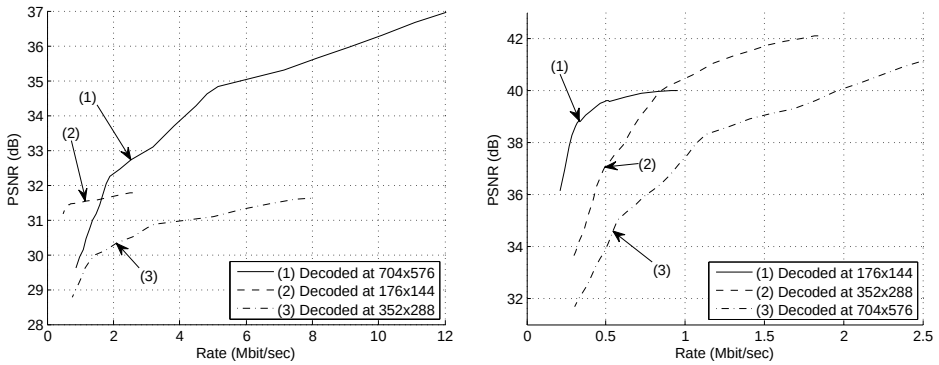
(b) Crew 704 × 576 pixels, 60 fps

Figure 4.16: Rate-distortion curves for the BDLD temporal configuration for the (a) City and (b) Crew sequences using full-search, ARPS-3 and HVSBM motion estimation techniques.

For the City sequence of Figure 4.16(a), ARPS-3 leads to a significantly lower coding performance when compared to the FS estimator. Further investigation has shown that ARPS-3 does not operate well on frames with large temporal distances (in the range 4-8 frame periods), as it is likely to converge to a local minimum. These local minima are abundant in the City sequence due to the high amount of repetitive patterns in the buildings. The HVSBM algorithm shows a higher performance than the FS estimator, due to the utilized variable block sizes, especially around occluding objects. However, the type of motion involved in the City sequence is uncommon in surveillance videos, as most cameras are stationary. More representative for surveillance applications is the Crew sequence of Figure 4.16(b). In this sequence, ARPS-3 gives slightly better performance than the FS estimator. The motion field found by the ARPS-3 is slightly smoother, which leads to less blocking artifacts and a more efficient motion-vector coding. Surprisingly, the HVSBM motion estimator performs poorly on the Crew sequence, due to excessive splitting of blocks when photographic flashes occur. This unnecessary splitting originates from luminance spikes triggering the block-splitting process. As a result, motion-vector coding cost becomes clearly higher, leading to multiple blocking artifacts.

Quality at reduced resolutions

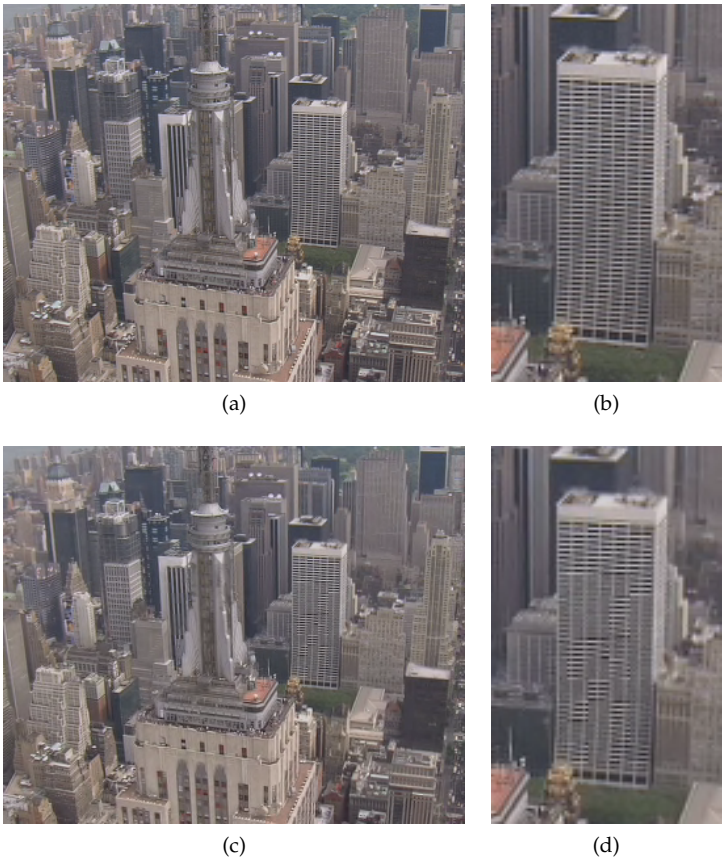
Due to the choice of the t+2D architecture, spatial aliasing will be evident in sequences decoded at lower resolutions. Although this aliasing reduces the PSNR significantly, the perceived quality only slightly degrades with some additional blurring effect. However, in special circumstances, these artifacts can become visually apparent, for example, in the highly detailed City sequence. Due to the high amount of fine structures, aliasing becomes clearly visible at one of the buildings in the background. Figure 4.18 shows Frame 64 of the City sequence. The original sequence is down-scaled to CIF resolution using the low-pass filter from the wavelet transform and depicted in Figure 4.18(a), with the building enlarged in Figure 4.18(b). The compressed sequence decoded at CIF resolution is depicted in Figure 4.18(c), with the building enlarged in Figure 4.18(d). In the compressed image, the complete coding system blurs certain parts of the aliasing pattern occurring in the original sequence. As a result, the PSNR drops significantly, as can be seen in the rate-distortion curve in Figure 4.17(a), see Curve (3). For QCIF resolution, most of the detailed structure in this building disappears during downsampling, which leads to less intrusive artifacts. This phenomenon is also noticeable from the slightly higher rate-distortion curve at this resolution, see Curve (2). Sequences that do not contain such highly detailed structures perform with more typical rate-distortion curves, of which an example is shown in Figure 4.17(b) for the CrossingA sequence.



(a) City 704 × 576 pixels, 60 fps

(b) CrossingA 704 × 576 pixels, 25 fps

Figure 4.17: Rate-distortion curves of (a) City and (b) CrossingA decoded at full (704 × 576 pixels) and reduced (352 × 288 and 176 × 144 pixels) resolutions.



(a) (b)

(c) (d)

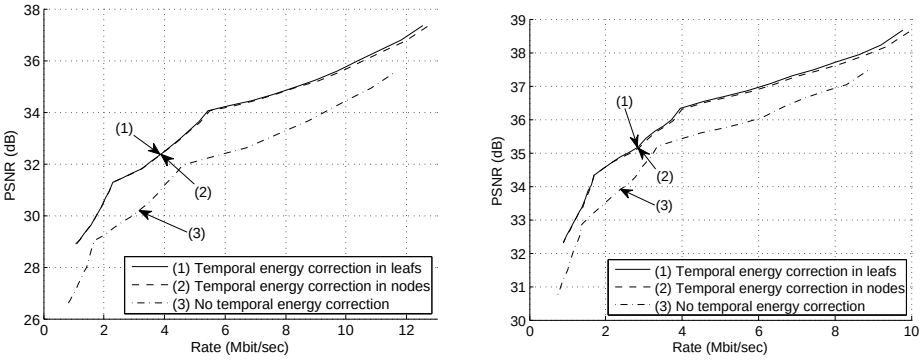
Figure 4.18: Spatial aliasing in the 4CIF City sequence decoded at CIF: (a) original frame, (b) detail of original frame, (c) compressed frame and (d) detail of compressed frame.

Experiments with the TEC and LCEF extensions

Temporal Energy Correction (TEC) and Low-Complexity Encoder Feedback (LCEF) are both extensions to the t+2D coding framework to alleviate some of its shortcomings. TEC isolates the temporal energy correction in the leafs of the temporal decomposition tree and reduces computational complexity, while LCEF introduces a low-complexity quality reduction for the first frame of the GOP, resulting in an improved average quality and reduced quality fluctuations. This section evaluates the improvements of these extensions.

Temporal Energy Correction (TEC)

The impact of applying TEC in the nodes or leafs of the tree is visualized in Figure 4.19 by means of Rate-Distortion (RD) curves. For simple rate control based on bit-plane truncation, either of the earlier described methods of TEC (see Section 4.4) shows significant improvements over a system without temporal energy correction. Furthermore, by placing the TEC in the leafs of the temporal lifting tree, a small quality improvement by up to 0.15 dB is observed, while significantly reducing the required amount of computations for TEC by about 43%, as only 16 full-frame energy-correction multiplications are required, instead of the 28 multiplications without TEC. For higher bit rates, the quality improvement between TEC in the nodes and leafs will be even larger because at these rates, the deliberate coding errors due to coefficient quantization reduce, so that the accumulated floating-point conversion errors become the dominant source of errors. By applying TEC in the leafs, this dominant source of errors is now significantly reduced.



(a) City 704 × 576 pixels (b) Crew 704 × 576 pixels

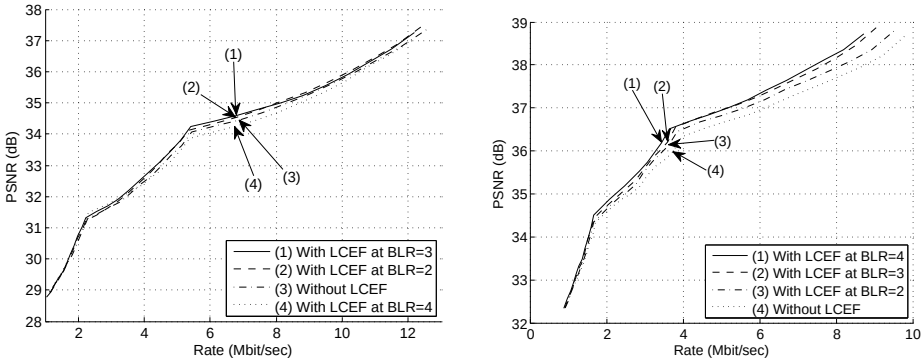
Figure 4.19: Temporal energy correction for (a) City and (b) Crew sequences.

Low-Complexity Encoder Feedback (LCEF)

Figure 4.20 portrays the effect of the LCEF on the average performance of the coding system by means of RD curves for the (a) City and (b) Crew sequences. The solid lines represent the coding performance without LCEF and the dotted/dashed lines represent the coding system with LCEF for different levels of Bit-Plane Reduction (BPR) applied to the Quality-Reduced Frame (QRF). For higher levels of BPR, the quality of the first frame becomes lower and eventually will introduce a growing quality drop. For lower levels of BPR, the performance converges to a coding system without LCEF. In the experiments, the optimal setting of BPR for the

QRF is 3 (BPR_{QRF}=3), which corresponds to a QRF in the quality and rate region of interest.

For improving clarity of the Figure 4.20, the results for LCEF with a $BPR_{QRF} > 4$ are omitted. For these high values of BPR_{QRF}, the visual quality of the first frame becomes so poor, that it will actually degrade the quality fluctuations within the GOP. However, these fluctuations cannot be observed from the traditional RD curves, where the associated small drop in average quality does not correspond with the perceived drop in visual quality.



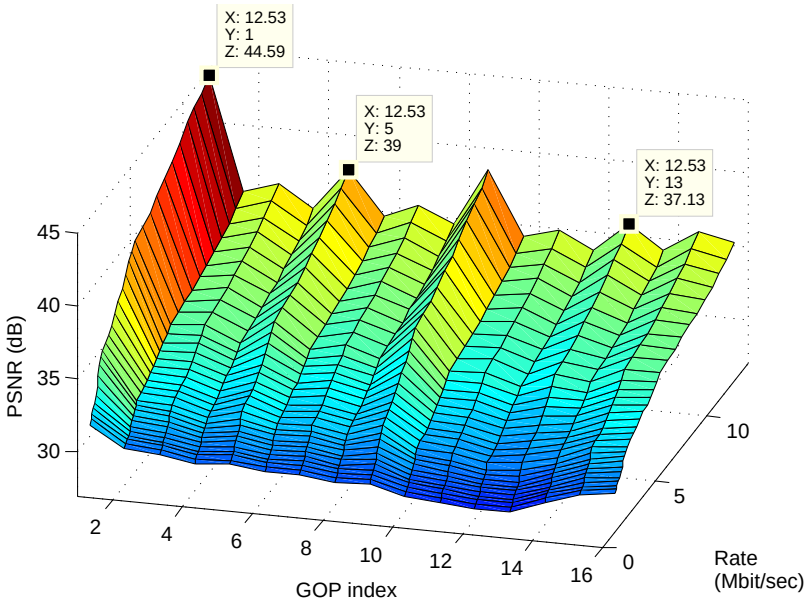
(a) City 704 × 576 pixels

(b) Crew 704 × 576 pixels

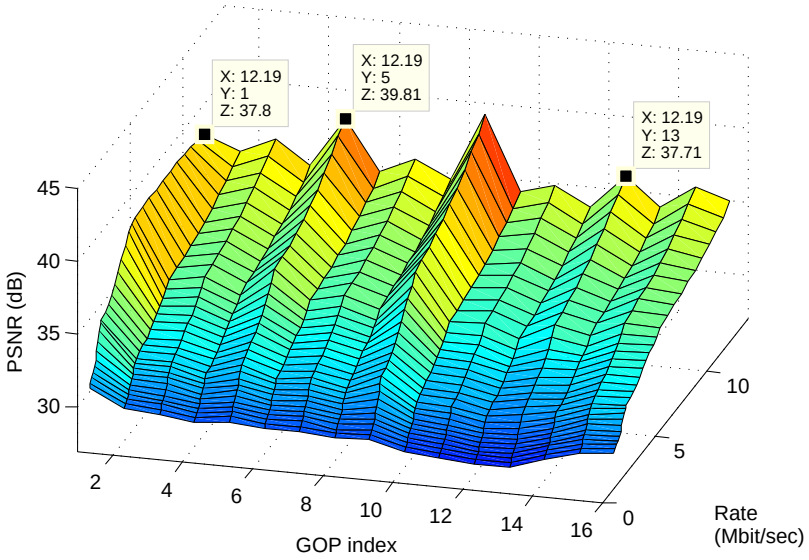
Figure 4.20: Effects of Low-Complexity Encoder Feedback at different Bit-Plane Reduction (BPR) levels within the QRF for the (a) City and (b) Crew sequences.

Traditional RD curves, as shown in the previous section, are not suitable for visualizing the quality fluctuations within a GOP. Due to the fixed GOP size of the SVC, repetitive quality fluctuations at a regular GOP interval (approx 0.5 s) can become particularly annoying, even when the average quality is acceptable due to the temporal low-frequency of the fluctuations. To visualize the quality fluctuations at the GOP interval, the average quality for all frames at a particular frame location within the GOP (the GOP index) is plotted in Figure 4.21. Hence, the index in the plot refers to all frames with that index, averaged over all GOPs. The rate indicated is the total rate for the complete stream, which is the same for all GOP indexes. By plotting the quality for various stream rates and GOP indexes, a 3D surface is obtained which displays the dynamic rate-distortion behavior within the GOP. By averaging the PSNR over all GOP indexes, the traditional RD curves of Figure 4.20 are obtained. Due to some limitation in our rate control, corresponding rate points are slightly different between Figure 4.21(a) and Figure 4.21(b).

Figure 4.21(a) and Figure 4.21(b) show the quality fluctuations within the GOP



(a) Original coding system



(b) Coding sytems with Low-Complexity Encoder Feedback (LCEF)

Figure 4.21: Within GOP quality fluctuations for (a) original coding system and (b) coding system with LCEF. The X-axis depicts the average rate in Mbit/sec, the Y-axis the frame index in the GOP and the Z-axis the PSNR in dB.

for the encoder without and with LCEF, respectively. An LCEF with $BPR_{QRF} = 3$ was chosen for this plot, with some points are highlighted, so that the reader can easily compare the figures. More detailed data from the figure can also be found in Table 4.5, where the quality between both encoders is compared at a bit rate of about 5.4 Mbps.

Table 4.5: Comparison of PSNR at each GOP index with and without LCEF.

GOP index	Without LCEF	With LCEF	Δ	Relative
1	39.60 dB	38.16 dB	-1.44 dB	-3.63 %
2	34.04 dB	34.13 dB	0.09 dB	0.26 %
3	34.31 dB	34.57 dB	0.26 dB	0.77 %
4	33.32 dB	33.56 dB	0.24 dB	0.71 %
5	35.00 dB	35.50 dB	0.51 dB	1.45 %
6	33.34 dB	33.63 dB	0.28 dB	0.85 %
7	33.85 dB	34.22 dB	0.37 dB	1.09 %
8	33.17 dB	33.49 dB	0.32 dB	0.96 %
9	35.48 dB	36.25 dB	0.77 dB	2.17 %
10	32.77 dB	33.06 dB	0.29 dB	0.88 %
11	32.91 dB	33.22 dB	0.31 dB	0.93 %
12	32.19 dB	32.44 dB	0.25 dB	0.76 %
13	33.23 dB	33.59 dB	0.37 dB	1.11 %
14	32.90 dB	33.08 dB	0.18 dB	0.54 %
15	34.02 dB	34.22 dB	0.20 dB	0.59 %
16	34.04 dB	34.11 dB	0.07 dB	0.18 %
Mean	34.01 dB	34.20 dB	0.19 dB	0.56 %
Variance	2.93	2.00	-0.93	-31.8 %
Rate	5.442 Mbps	5.408 Mbps	-34 Kbps	-0.62 %

From both Figure 4.21 and Table 4.5, it can be clearly observed that when LCEF is utilized, the quality of the first frame of the GOP is reduced, while the remaining frames gain in quality. As a result, the quality is more consistent, which is derived from a flatter surface in Figure 4.21(b) and a reduced variance of the PSNR within the GOP in Table 4.5, going from 2.93 to 2.00. The LCEF extension reduces the largest peak in the GOP to a level closer to the remaining frames, thereby significantly reducing quality fluctuations and increasing the average quality level for virtually all frames.

Quality verification versus state-of-the-art

Scalability can be added to the well-known H.264 coding system through the SVC extension. This extension is rarely supported in commercially deployed systems, but does provide similar functionality to the proposed scalable video coding system. The H.264 SVC coder is significantly more complex and rich with features, which hampers implementation on embedded systems. This section evaluates the performance of the low-complexity coding system, so that its performance can be compared to the H.264 standard.

For this experiment, the proposed coding system is compared to the H.264 SVC CE2.1 coder (which was current at the time of experimentation) for the Crew test sequence. This sequence is chosen, since its motion characteristics are quite similar to those occurring in surveillance type videos: a camera at a fixed location following objects moving in the scene. Within the proposed SVC, the well-known CDF 9/7 filter bank is used for the spatial wavelet transform, in combination with a 5-level dyadic wavelet decomposition. The GOP size is 16 frames, with an adaptive 4-level temporal wavelet transform. Motion is estimated up to 1/4 pixel accuracy, using the ARPS-3 block-based motion estimator with a fixed block size of 16×16 pixels. Rate control is performed by simply reducing the number of decoded bit planes and for a more accurate control of the bit rate, the last bit plane is partially truncated. The temporal low-pass frame is encoded with one additional bit plane, compared to the other temporal high-pass frames. Entropy coding of the wavelet coefficients is performed with the TSSP coder of Chapter 3 and the motion vectors are encoded using a Reversible Variable Length Coding (RVLC) technique. Although used in H.264, arithmetic coding is not employed in the proposed SVC for complexity reasons.

From the curves in Figure 4.22(a), it shows that the proposed SVC obtains a slightly lower PSNR than H.264 SVC (0-1 dB) at the same bit rate. Regarding visual quality, the proposed SVC in Figure 4.22(d) provides more visual details at the cost of being slightly more noisy than H.264 SVC shown in Figure 4.22(c). However, this coding behaviour is even preferable for surveillance applications.

4.6 Conclusions

This chapter describes the investigated trade-off between complexity and performance in scalable wavelet coders aiming at video surveillance applications. The first part of the chapter evaluates the merits of several coding architectures, both predictive (conventional and scalable) and 3D subband coders (t+2D, 2D+t and 2D+t+2D). Various performance metrics have been addressed, such as: inherent

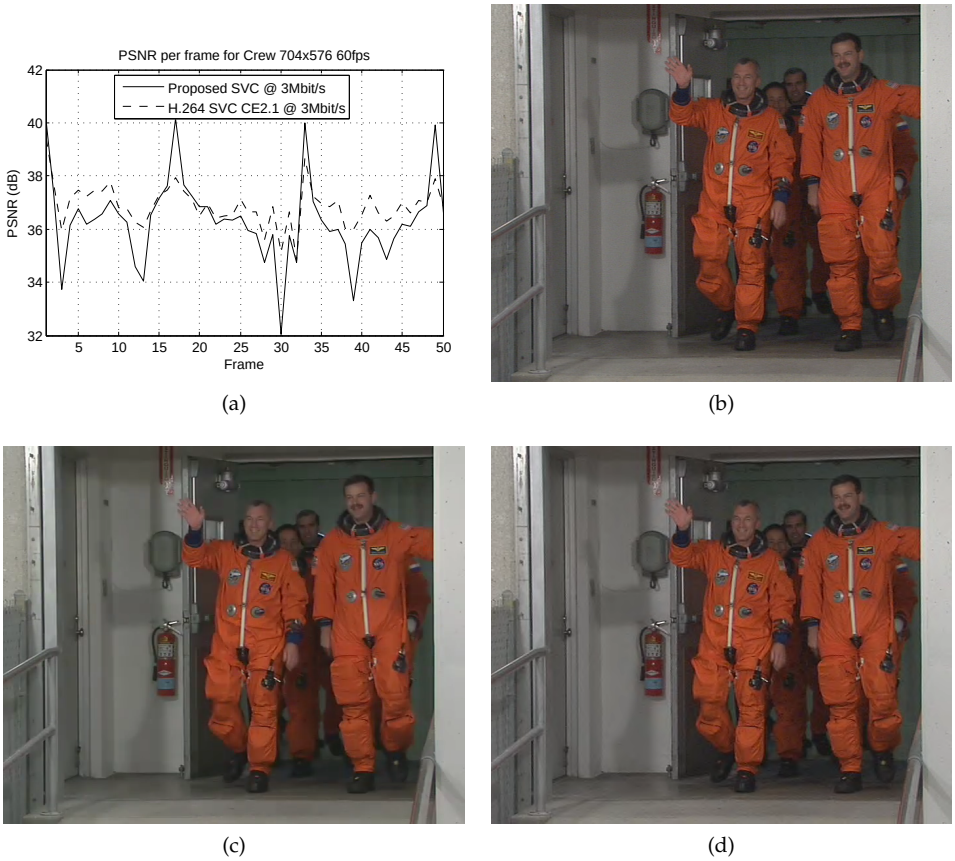


Figure 4.22: Comparison of H.264 SVC and our proposed SVC for the Crew sequence @ 3Mbit/s: (a) PSNR per frame, (b) original Frame 32, (c) H.264 SVC Frame 32 and (d) proposed SVC Frame 32.

and motion complexity, coding performance, scalability in quality, resolution and frame rate and surveillance domain-specific features such as end-to-end delay, trick play and random access. Based on this evaluation, we have selected the t+2D framework for its inherent low complexity, good scalability options for resolution, quality and temporal dimensions and support for domain-specific features. The only downside is the possibility of aliasing artifacts at reduced resolutions and detailed patterns. However, from a domain perspective, this shortcoming is acceptable and less important than the other performance points.

Subsequently, the adopted t+2D framework is explored with respect to various

temporal configurations, involving both complexity parameters and performance metrics. The proposed BDLD temporal configuration has been compared to several other UMCTF configurations. We have found that the BDLD configuration features moderate computation complexity and low end-to-end delay, while still achieving sufficiently high quality. Balancing such factors, we have adopted the BDLD for detailed evaluation. During this investigation of different temporal coding structures, it has been shown also that the update steps play a significant role in the perceived quality due to specific coding artifacts. In the architecture, the quality of the LLLL band can be improved for long-term storage in surveillance by consciously removing the update steps.

The analysis of the t+2D coding framework has revealed two shortcomings. The first involves inefficiencies in the multi-level floating-point energy correction of the temporal wavelet transform. The second relates to strong fluctuations of the quality of frames within a GOP. Two extensions have been proposed to alleviate these shortcomings, in which the system deviates from the standard t+2D coding structure: Temporal Energy Correction (TEC) and Low-Complexity Encoder Feedback (LCEF).

TEC isolates the temporal energy correction in the leafs of the temporal decomposition tree and significantly reduces computational complexity by about 43%, while simultaneously memory bandwidth is reduced even further, for both the encoder and the decoder. Furthermore, the accumulating quantization errors from successive fixed/floating-point conversions are removed, which additionally leads to a small quality increase of up to 0.15 dB.

LCEF introduces a low-complexity quality reduction for the first frame of the GOP, for which decoder modifications are not required. Using this technique, the average quality of the frames within a GOP is improved and quality fluctuations are significantly reduced. This is expressed by a reduction of the variance of the PSNR by 30%. Furthermore, the average quality is improved by 0.56 dB. Since both phenomena jointly occur, this results in a significantly higher perceived quality.

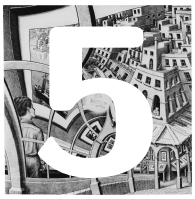
With respect to coding quality in general, the coding performance of the proposed SVC at full resolution is close to H.264 SVC (within 1 dB for surveillance type video) and at lower resolutions sufficient for video surveillance applications, albeit with a much lower complexity. The most dominant complexity reductions are found in the non-hierarchical motion model, the straightforward t+2D architecture and the computationally efficient TSSP entropy coding scheme without expensive arithmetic coding. Perceptually, the codec retains important visual de-

tails. Scalability can be accurately controlled with fine-grained scalability in quality, while simultaneously supporting multiple options for resolution and framerate scalability. Aliasing artifacts may occur for lower resolutions, but these are acceptable for video surveillance applications.

This chapter has only briefly considered the performance of various motion estimation techniques. However, none of these motion estimation techniques have been developed for the purpose of scalable video coding. Therefore, Chapter 5 develops a novel motion estimator, which is dedicated to scalable video coding and efficient hardware implementations.

At this point, a complete scalable video coding system has been developed with various scalability options, low complexity and suitability for embedded systems implementation. This was obtained through the use of integer wavelet transforms (Section 2.3), the proposed TSSP entropy coder (Section 3.2) and the BDLD temporal configuration (Section 4.3). The disadvantage of the proposed proprietary architecture is that it is specific and not adhering to existing video compression standards. The development of an embedded coding system for a specific application area is relatively expensive, whereas adherence to existing standards can give a cost benefit on the longer term if this standard is widely adopted.

With respect to comparing the developed system with the H.264 SVC compression standard, an interesting observation can be made about the temporal scalability in both systems. In this chapter, the created temporal hierarchical tree (BDLD) is the actual result from various research explorations which initially use complete temporal wavelet filters. The equivalent development in the standardization of H.264 SVC has resulted in a hierarchical use of B-frames which exploits the cascaded dependencies of such frames. From a bird's eye view, both approaches are conceptually nearly identical, although the development paths have been very different. The fact that both approaches are conceptually similar, gives further motivation that the correct temporal coding structure is adopted.

CHAPTER 

Motion estimation for scalable video coding

> Time has been transformed, and we have changed; it has advanced and set us in motion; it has unveiled its face, inspiring us with bewilderment and exhilaration.
>
> KAHLIL GIBRAN

Abstract

This chapter concentrates on motion estimation specifically for Scalable Video Coding (SVC). In SVC, motion is estimated with frames at varying temporal distances and new methods of motion-vector propagation are necessary. Furthermore, hardware accelerators in modern computing architectures shift the complexity bottleneck from SAD computations to the required memory bandwidth. Therefore, a novel motion estimation algorithm is presented in this chapter, that is explicitly designed for SVC, which exploits hardware acceleration features of modern computing architectures. Fast processing is not only achieved by employing a few candidate motion vectors, but also by tuning the algorithm such that it starts with two candidate vectors and then checks the candidate positions with a surrounding dense pattern arranged in a Parallelogram-Shaped Scanning (PSS) form. As an advantage, this results in a fixed computational load, while the two separate vector candidates allow parallel fetching of data and SAD calculations. The proposal is compared with other motion estimation algorithms, including the state-of-the-art EPZS motion estimation. We have found that the proposed motion estimation outperforms all other fast motion estimation algorithms in the SVC framework and even approaches the performance of a full-search motion estimation.

5.1 Introduction

Various preceding chapters have contributed to the complete proposed scalable video coding system, as portrayed by Figure 5.1. Chapter 3 has contributed by developing an efficient TSSP entropy codec. Chapter 4 has provided a scalable coding framework based on temporal wavelet decomposition and proposed the BiDirectional configuration with Low Delay, abbreviated as BDLD. Up to this point, for the t+2D architecture, a detailed evaluation and optimization of the involved Motion Estimation (ME) algorithm has not yet been addressed. It is plausible that when considering all desired system aspects of surveillance, an algorithmic study of motion estimation is expected to lead to a motion estimation algorithm fitting to the scalable video coding system and suited for embedded system implementation.

For the proposed SVC as studied in the previous chapters, ME algorithms have been applied which are also commonly used in other video coding systems. Since most video coding systems are non-scalable, the applied ME algorithms were never specifically designed for this application. Therefore, this chapter concentrates on motion estimation for scalable video coding and also considers the latest developments in computing architectures as a guideline for algorithmic design.

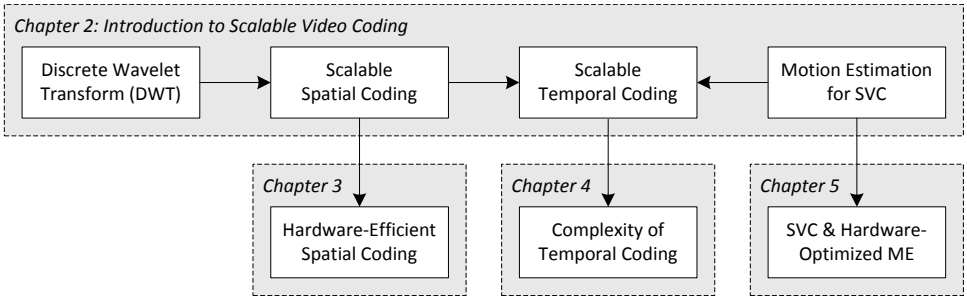


Figure 5.1: Positioning of this chapter in the research scope of this thesis.

ME algorithms are used in state-of-the-art video coding systems, such as the well-known H.264/AVC [15] and novel scalable video coding systems like MC-EZBC [82]. Various types of ME algorithms have been proposed over the past 30 years. Full-search or exhaustive-search ME algorithms check all possible motion vectors within a certain search window and return the vector with the lowest Sum of Absolute Differences (SAD). Computational complexity of these ME algorithms is quadratic with the search window size, so that multiple fast ME algorithms have been proposed. Many fast ME algorithms employ a multi-stage approach, such as TSS (Three-Step-Search) [100], ARPS-3 (Advanced Rood Pattern Search) [99],

PMVFAST [101] and EPZS (Enhanced Predictive Zonal Search) [102]. The common principle is that a set of potential vectors is scanned according to a predetermined pattern (e.g. large diamond), from which the best vector is selected as the input for the next stage, being based on a somewhat modified pattern (e.g. small diamond). The algorithm continues for a few iterations, until the optimal vector is found. Some ME algorithms utilize early-stop mechanisms to further save on calculations [101] [102] [103]. The main purpose of these fast algorithms is to reduce the number of SAD calculations, which occupy a large part of the complexity in sequential software implementations. However, parallel computing architectures calculate the SAD of a block significantly faster than sequential processors, so that the main bottleneck shifts from computations to memory bandwidth. Due to the sequential nature of many fast ME algorithms, they do not map efficiently onto parallel computing architectures. Furthermore, most ME algorithms were designed for predictive video coding systems, employing straightforward frame-by-frame motion estimation. However, SVCs feature also bidirectional processing, where the motion is estimated at various temporal distances (e.g. 1, 2, 4 and 8 frames apart) in forward and backward direction.

In conclusion, typical motion estimation algorithms used for video coding were neither designed for scalable video coding, nor designed for parallel computing implementations. Therefore, this chapter aims at designing an efficient ME algorithm that satisfied both requirements in one design. To this end, we propose a novel ME algorithm, named HPPS (Highly Parallel Predictive Search), with three distinct features: motion estimation for SVC, suitability for parallel computing architectures and featuring a fixed computational load.

This chapter is organized as follows. Section 5.2 briefly relates motion estimation to the framework of temporal decomposition as used in the proposed SVC. It also discusses three commonly used ME algorithms, which have influenced the design of the new HPPS ME algorithm. Section 5.3 presents this novel HPPS ME algorithm in detail. Section 5.4 deals with the experimental results and compares the proposed ME algorithm with other algorithms from literature. Finally, Section 5.5 concludes this chapter.

5.2 Motion estimation in the temporal tree

BDLD temporal configuration

For our surveillance application, the BDLD (BiDirectional Low Delay) configuration was adopted in Chapter 4. This configuration uses bidirectional prediction at the two lowest levels (P_2U_0) and single prediction at the two highest levels (P_1U_0),

in order to reduce the coding delay. Figure 5.2 shows the BDLD temporal configuration for the proposed SVC with a four-level temporal configuration. Bidirectional Motion Estimation (ME) can be observed at the top layers, while ME over large temporal distances can be observed at the bottom layers. Motion estimation that is performed between frames with large temporal distances requires a significantly expanded search region, thereby adding complexity in a quadratic order with larger search distances.

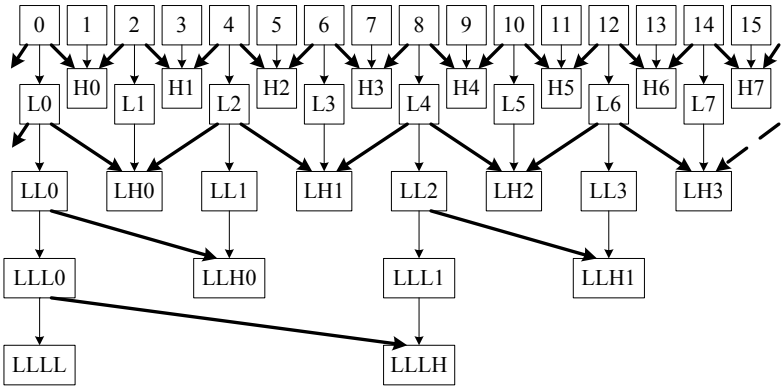


Figure 5.2: BDLD temporal configuration for the proposed SVC with a four-level temporal transform. Bold arrows represent the lifting steps that include ME and compensation.

Figure 5.2 also shows the complex motion estimation process in SVCs. ME is performed within the lifting steps, which is indicated in the figure by the bold arrows. Each of these bold arrows represent a shorthand notation for the complex process of motion estimation, motion compensation and temporal filtering using the lifting framework. The figure explaining the internals of these bold arrows is repeated from the previous chapter in Figure 5.3, depicting the internals of the bidirectional prediction step from the top of Figure 5.2. For a more detailed description of the BDLD temporal configuration, see Chapter 4.

Commonly used ME algorithms

This section discusses three commonly used ME algorithms which have become popular in video coding and frame-rate conversion. They form the basis for the later proposed HPPS ME algorithm. The first algorithm, called ARPS-3 (Advanced Rood Pattern Search) [99], is a simple and fast multi-stage ME. The second algorithm, EPZS (Enhanced Predictive Zonal Search) [102] is based on PMVFAST (Predictive Motion Vector Field Adaptive Search Technique) [101], which enhances

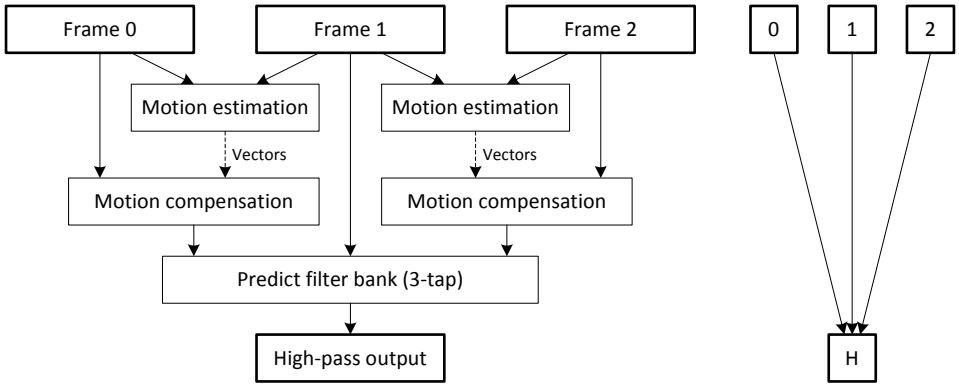


Figure 5.3: Temporal filtering in the t+2D coding architecture using 5/3 wavelet filters. (a) Full structure and (b) shorthand notation with motion estimation/compensation structure represented by the diagonal arrows, with filtering implicitly part of the processing.

the multi-stage design with early stop criteria and utilizes previously obtained motion-vector fields. The third, 3DRS (3-Dimensional Recursive Search) [104], is a very fast ME which is applied for frame-rate conversion, as it provides smooth motion-vector fields and enables a simple implementation in hardware.

Block similarity measure

All motion estimation algorithms from this chapter use the Sum of Absolute Differences (SAD) as the measure of similarity between two blocks. The SAD is defined as follows:

$$SAD_{n,m}(x,y,dx,dy) = \sum_{(i,j)\in B} \mid I_n(x+i,y+j) - I_m(x+i+dx,y+j+dy) \mid, \quad (5.1)$$

with (i,j) being the pixel positions within block B, which is typically a 16×16 pixel block. The signal component value at pixel location (i,j) in image I_n is denoted by $I_n(i,j)$. The reference location of block B is denoted by pixel position (x,y) and the tested motion vector is represented by the displacement (dx,dy). In shorthand notation, the image indexes are omitted and the SAD is written by $SAD(x,y,dx,dy)$. Besides the SAD, other similarity measures exist, but are hardly used in practice because of the effectiveness, computational simplicity and wide-spread acceptance of the SAD measure.

ARPS-3

The ARPS-3 ME reduces the number of possible motion-vectors candidates compared to the full-search algorithm. ARPS-3 centers its search around a predicted motion vector, after which several motion-vector values are tested, based on the motion vectors in neighbouring blocks. The search is initiated using a rood pattern, after which a smaller pattern is iteratively used.

Figure 5.4 shows a grid of possible integer motion-vector positions. Figure 5.4(a) shows the location of the tested vector positions around the predicted motion-vector center point for the initial search. The points are derived from motion vectors of the neighbouring blocks, using the assumption that the expected variations among motion vectors is similar to the variations of the motion vectors of neighbouring blocks. The arms of the rood, points MV_1 to MV_4, are directly based on the maximum and minimum values of the surrounding motion vectors (therefore they vary in length in the figure). The origin point MV_5, representing the (0,0) vector, is also included for robustness of the algorithm.

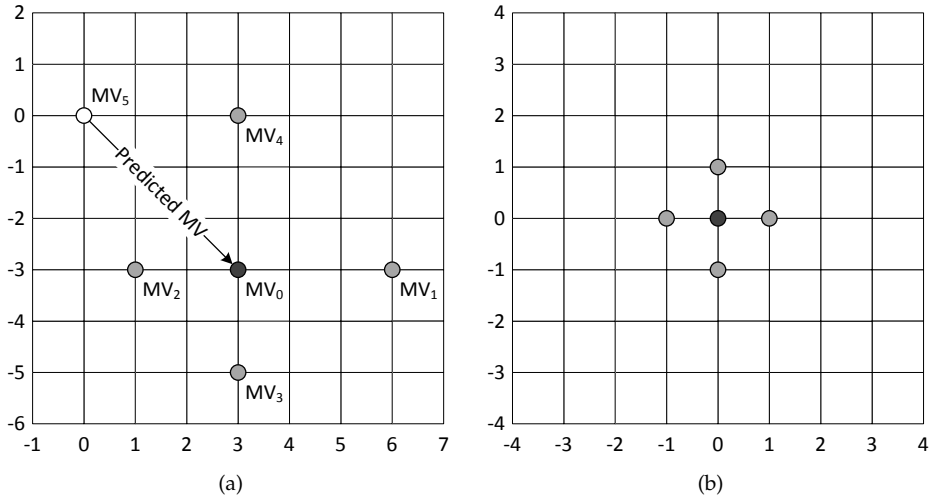


Figure 5.4: Search patterns in ARPS-3: (a) Rood pattern, adapted from [99] and (b) Unity Rood Pattern (URP), adapted from [105] and [106].

After the initial search with the rood pattern, a smaller pattern is used iteratively to converge to a final position. This smaller pattern is also known as the Unity Rood Pattern (URP) [105], or the small diamond pattern [106] and is shown in Figure 5.4(b).

PMVFAST and EPZS

EPZS [102] is an improvement version of the PMVFAST [101] algorithm, based on enhancements of the motion-vector prediction process and the early stopping criteria. Due to the enhanced reliability of the predictor, only a single checking pattern is used, either a small 3×3 diamond or a 3×3 square, depending on the user requirements.

Similar to ARPS-3, PMVFAST and EPZS use previously obtained motion vectors from both previously calculated fields and spatial neighbours. PMVFAST first evaluates the median predictor, which is also used as a reference for motion-vector encoding. After testing the median, a set of vectors is evaluated involving the (0,0) vector, the vectors from three adjacent blocks in the current frame (left, top and top-right) and the co-located block in the previous frame, amounting to a total of six predictors. The median predictor represents *subset A*, while the others are grouped in *subset B*.

PMVFAST also uses early stop criteria based on thresholds. The early stop after testing the median position is based on a fixed threshold, whereas for the other predictors, it is adaptively based on the minimum SAD of the adjacent blocks or the co-located block.

EPZS expands on PMVFAST by including another set of predictors using previous frames. The first is called the accelerator motion vector, that utilizes the motion vectors of the co-located blocks at time $t-1$ and $t-2$, which is included in Equation (5.2), as follows:

$$\overrightarrow{MV}_{AccelPredictor} = \overrightarrow{MV}_{t-1} + (\overrightarrow{MV}_{t-1} - \overrightarrow{MV}_{t-2}), \tag{5.2}$$

where $\overrightarrow{MV}_{AccelPredictor}$ denotes the predicted accelerator motion vector and $\overrightarrow{MV}_{t-1}$ and $\overrightarrow{MV}_{t-2}$ the motion vectors of the co-located blocks at time $t-1$ and $t-2$.

For objects with motion displacements in the order of the size of the motion-vector blocks, the co-located block is not representing the motion in the current frame at the object boundary. Therefore, the motion vectors of the four adjacent blocks (left, right, top and bottom) to the co-located block in the previous frame are included as candidate motion vectors as well, bringing the additional temporal predictors in EPZS to five, which are grouped in *subset C*.

Early stop criteria are expanded in EPZS as follows: a fixed threshold (preset at a value of 256) is used for *subset A* and an adaptive threshold is used after testing all predictors in *subsets B*[1] and *C*. The value of the adaptive threshold in EPZS is

[1]The explanation of the EPZS algorithm and the labeling of subsets is according to the literature [102], but should not be confused with the previous discussion on block *B* in the SAD computation.

based on the minimum SAD of the tested predictors so far, combined according to Equation (5.3), which states that:

$$T_{Adaptive} = a \cdot min(SAD_1, SAD_2, ..., SAD_N) + b, \tag{5.3}$$

with $a = 1.2$ and $b = 128$. By combining the SAD of the spatial and temporal predictors, a pre-mature early stop of the algorithm can be prevented successfully. If no early stop conditions occur, the motion estimation computation continues, using a refinement pattern similar to ARPS-3, albeit that this computation is extended for a limited and predefined amount of iterations.

3DRS

3DRS was proposed primarily as an efficient ME algorithm for frame-rate conversion in high-end television sets. Since frame-rate conversion in TVs has to be performed in real-time, the 3DRS motion estimation was designed for low-cost hardware implementation with respect to the amount of computations. The 3DRS algorithm attempts to estimate a smooth true-motion field, suitable for frame-rate conversion without serious outliers, while still supporting sudden changes in motion, resulting from boundaries between differently moving objects.

Despite its usage for a specific area, we have adopted this algorithm for further study because the system has proven to be a valuable concept from which new extensions can be made (e.g. a 3D TV extension can be found in [107]).

Similar to the previously described ME algorithms, 3DRS uses predictors based on the spatial neighbourhood and the previously obtained motion vectors, while also additionally testing the (0,0) vector. A possible candidate structure for 3DRS is shown in Figure 5.5.

3DRS also includes a random update vector that facilitates the sampling of a different motion field in the neighbourhood. For that random update vector, a limited set of vectors is defined, of which one is randomly chosen and added to a selection of the pre-defined candidates. A possible set of random update vectors is shown below, giving

$$U = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \begin{pmatrix} -3 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ -2 \end{pmatrix} \right\}. \tag{5.4}$$

Motivation for a novel ME algorithm

The discussed ME algorithms are all based on the traditional assumption that reducing the number of SAD operations is the best way to reduce the computational

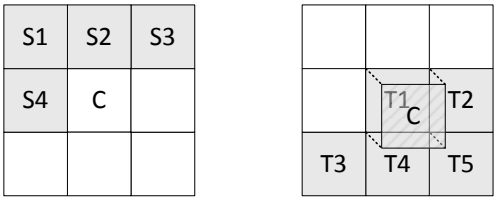


Figure 5.5: Possible motion-vector candidates of 3DRS: (a) spatial candidates at positions S1–S4, (b) temporal candidates at positions T1–T5. The considered block for which the candidate vectors apply is indicated by position C. Block position C is dotted in sub-figure (b) to indicate that the candidate vectors at positions T1–T5 are computed already earlier, where block T1 is co-located with block C.

complexity, as this is the most computationally expensive part of the ME algorithm. While this is valid for traditional 'sequential' processing cores, developments in parallel processing and dedicated hardware accelerators for multimedia computing processors are changing the landscape. New processor architectures have emerged, featuring dedicated execution units to accelerate the block-based SAD calculation from thousands of cycles to a single cycle. Furthermore, these architectures usually also include dedicated memory execution units to shift all pixels in the motion estimation block, which can be utilized to move to the next candidate region efficiently. With these new execution units, the computational bottleneck has shifted from the SAD calculation and search process to the available memory bandwidth. This results in different requirements for designing motion estimation algorithms.

As a consequence of the previous discussion, the ME algorithm should be efficiently mapped and executed on a multimedia computing system, but should also support scalable video coding systems which have a hierarchical temporal structure, requiring typically ME between frames with a large temporal distance. Furthermore, for real-time systems such as a live video encoder, it is preferable that the ME algorithm has a fixed computational complexity, so that hard deadlines can be guaranteed.

In conclusion, the following three aspects should be addressed in the design of the algorithm: (1) the ME should fit to SVC and support bidirectional and temporal predictive coding hierarchies, (2) consideration of the available memory bandwidth instead of SAD calculations and (3) the ME algorithm should match with the adopted computing architectures for execution. These aspects have been used as a starting point for designing a novel ME algorithm.

5.3 Highly Parallel Predictive Search (HPPS)

The following sections discuss the novel HPPS ME algorithm in more detail. Section 5.3 first gives an overview of the novel HPPS algorithm and the following sections describe HPPS in more detail. Section 5.3 presents the candidate generation, Section 5.3 the temporal candidate input in an SVC and Section 5.3 presents a special parallelogram-shaped scanning pattern. Finally, Section 5.3 presents cost biasing.

Overview of HPPS approach

In the previous section, the motivations for a novel ME algorithm were explained: suitability for SVCs, the shift in the computational bottleneck from SAD calculations to memory bandwidth and design-phase consideration for mapping on predefined target architectures. For the design of HPPS, three successful concepts of the previously discussed ME algorithms are adopted.

The first aspect involves the use of square blocks for ME algorithms. The square block as used in many ME algorithms for video coding, is attractive since it matches the block-based processing of the Discrete-Cosine-Transform (DCT) with a block-based ME process. However, in wavelet-based systems, the transform is typically applied on full-image basis. This makes motion estimation without using blocks a natural choice, in order to avoid blocking artifacts commonly associated with block-based coding systems. Novel ME proposals exist where deformable mesh triangles are used, which also allows for changes in object size. Although the motion compensation quality of such algorithms is very high, the deformability of the mesh triangles adds additional search dimensions to the ME process, which increases its complexity exponentially. Furthermore, block-based ME algorithms have been broadly accepted which has resulted in readily available special hardware accelerators for modern hardware computing platforms. For these reasons, the block-based processing for the novel ME algorithm is still adopted.

The second aspect is related to the choice of a block similarity metric. For measuring block similarity, the Sum of Absolute Difference (SAD) calculation is commonly used in ME because of its computational simplicity. In experiments, we have explored a novel block-difference calculation, based on the wavelet transform. This novel difference calculation uses a multi-scale approach similar to the spatial wavelet transform, but unfortunately, this does not result in improvements in the coding quality. Moreover, the wide application of the SAD calculation has also led to the availability of special hardware accelerators in modern computing platforms. For these reasons, the SAD calculation is again used for block-difference

calculations in the novel ME algorithm.

The third aspect addresses the use of predictive motion vectors. It has been discussed and observed that a good predictor is very important for the performance of ME algorithms. This predictor can be estimated from both spatial and temporal neighbouring blocks, while the median operator is effective for selecting a reliable predictor from those neighboring blocks. Therefore, the concept of spatial and temporal predictors is adopted for the novel ME algorithm. However, the search strategy is modified, in order to optimize it for architectures with dedicated SAD calculation and hardware blocks specialized in data shifting.

A schematic overview of the proposed HPPS ME is given in Figure 5.6. The three independent processing paths can be clearly observed: one path for checking the $(0, 0)$ motion vector (also called zero motion vector) at the top of the figure, one depending on a spatial predictor at the middle of the figure and one path depending on a temporal predictor at the bottom. The final motion vector equals the motion vector with the lowest SAD of the three paths. Within both predictive paths, a single predictor is computed based on a selection of candidates from the input motion-vector field, which is discussed in Section 5.3. The spatial predictor path utilizes previously calculated motion vectors from the current frame, indicated in the figure by the feedback loop. For the temporal predictor, the employed input motion-vector field originates from previous ME actions in the hierarchical tree. For SVC, this motion-vector propagation is not trivial and Section 5.3 describes two methods for propagating candidates in the temporal hierarchical tree of the proposed SVC, which is indicated in Figure 5.2. For each of the motion-vector predictors computed, a dense search is performed around the considered predictor using the Parallelogram-Shaped Scanning (PSS) pattern, which is discussed in Section 5.3. The results of this dense search can be modified by applying cost biasing, which is further explained in Section 5.3. For the zero motion-vector path, the dense search and biasing of vector cost are optional, as indicated in Figure 5.6 by the diagonal hatching of the optional blocks.

Calculation of spatial and temporal predictors

HPPS uses two motion-vector predictors to initiate a dense search: one vector derived from the spatial neighborhood of motion vectors and one vector derived from the temporal neighborhood. The temporal neighborhood can be chosen in relatively free form, allowing the fetching of image data of various blocks for the temporal candidate independent of the calculation of spatial neighborhood predictions. Alternatively, the image data for the spatial candidate can be fetched during the temporal neighborhood calculation, thereby enabling continuous parallel data

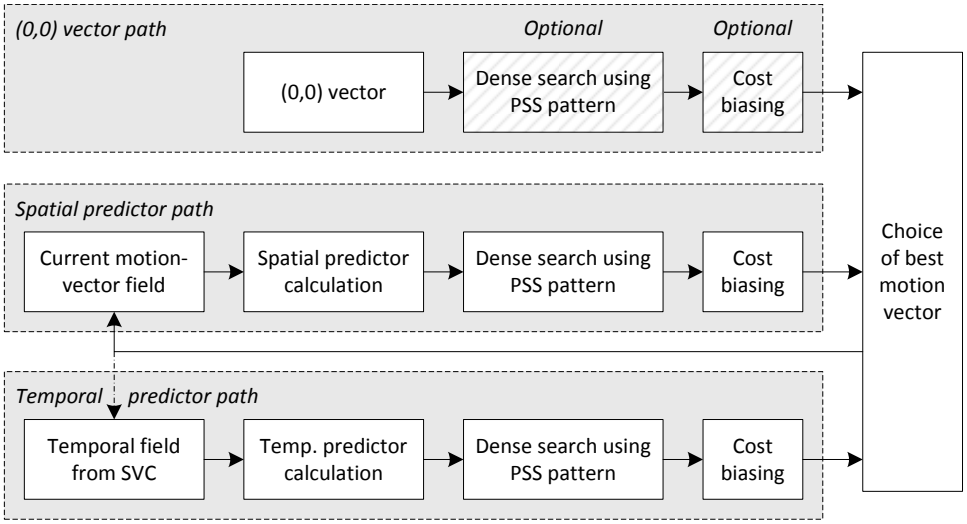


Figure 5.6: Schematic overview of the Highly Parallel Predictive Search (HPPS) ME.

fetching and SAD calculations, see Figure 5.6.

Around both the spatial and the temporal predictors, HPPS performs a modified full search. This differs from TSS and ARPS-3, in the sense that it does not employ an iterative process for the candidate-vector evaluation. It also differs from 3DRS [104], since it refines the motion vector by exploring a search area around each predictor.

Figure 5.7 depicts the blocks for the motion-vector prediction, showing (a) the spatial and (b) the temporal neighborhood of motion vectors, with C indicating the current motion vector predictor to be determined. Spatial positions of the spatial candidates are represented by S1–S3 and the temporal candidates by T1–T4. Spatial candidates are derived from the current frame, while temporal candidates originate from a temporal candidate frame. The spatial motion-vector prediction ($\overrightarrow{MV}_{SpatPred}$) and temporal motion-vector prediction ($\overrightarrow{MV}_{TempPred}$) are derived from these positions according to Equations (5.5) and (5.6). Both predictors employ the median operator for vector selection, which was shown by Tourapis *et al.* [101] [102] to be a good predictor for the current motion vector. Hence, both predictors are specified by

$$\overrightarrow{MV}_{SpatPred}(C_S) = \text{median}(\overrightarrow{MV}(S1), \overrightarrow{MV}(S2), \overrightarrow{MV}(S3)), \quad (5.5)$$

$$\overrightarrow{MV}_{TempPred}(C_T) = \text{median}(\overrightarrow{MV}(T1), \overrightarrow{MV}(T2), \overrightarrow{MV}(T3), \overrightarrow{MV}(T4)), \quad (5.6)$$

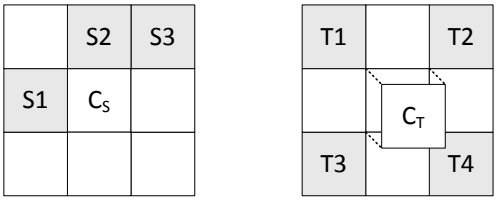


Figure 5.7: Motion-vector candidates for spatial and temporal prediction in HPPS: (a) spatial candidates at positions S1–S3, (b) temporal candidates at positions T1–T5 with the considered block for which the candidate vectors apply is indicated by position C_S and C_T, respectively. Block position C_T is shown lifted to indicate that the candidate vectors at positions T1–T4 are computed already earlier.

with $\overrightarrow{MV}(P)$ the candidate motion vector at location P. The median operator in these equations operates individually on the x and y components of each motion vector. If not all motion vectors are available (e.g. around image edges), missing candidates are omitted from the median calculation. Temporal relationships in the BDLD configuration of the SVC framework are addressed in more detail in Section 5.3.

Origin of temporal vector candidates in SVC

This section provides the input for the temporal prediction processing of the previous section. Since a region around each motion-vector predictor is scanned, the propagation of temporal vector candidates can be implemented in a simple way. This propagation of vector candidates is visualized in Figure 5.8 for a four-level SVC with a BDLD temporal configuration. For the first level with bidirectional ME, the motion-vector field is simply reversed, while for higher levels, the motion-vector field is scaled by a factor two from levels below. These simple techniques are sufficient for providing temporal candidates at each level of the decomposition, as these candidates are resulting in a temporal predictor (See Section 5.3.2), around which a dense search is performed.

The temporal vector candidate propagation of Figure 5.8 can be split in two categories: intra-level propagation and inter-level propagation. The temporal intra-level and inter-level motion vectors $\overrightarrow{MV}$ are defined through Equations (5.7) and

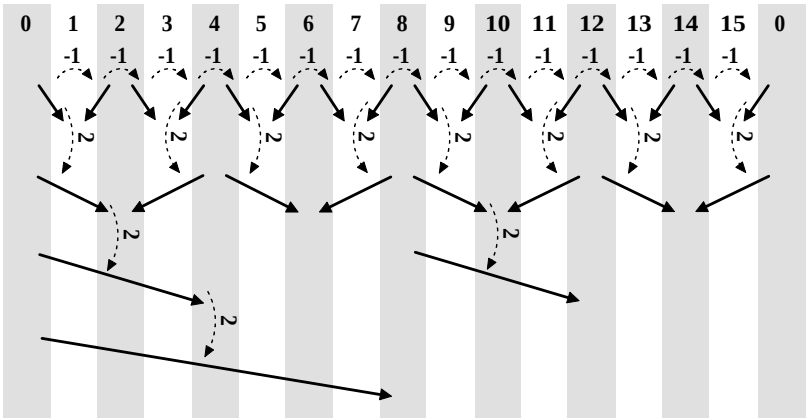


Figure 5.8: Temporal candidate propagation in a four-level SVC with every vertical stripe (columns) representing a frame, where the top numbers indicating the frame index in the GOP. Normal arrows indicate motion estimation calculations with candidate vectors represented by the dashed arrows. The number next to the dashed arrow indicates the multiplication factor of the motion-vector field.

(5.8), respectively, giving

$$\overrightarrow{MV}_{TempCand}(P,0) = -\overrightarrow{MV}_{Prev}(P,0), \tag{5.7}$$

$$\overrightarrow{MV}_{TempCand}(P,l) = 2 * \overrightarrow{MV}_{Prev}(P,l-1) \qquad \text{for } l > 0, \tag{5.8}$$

with $\overrightarrow{MV}_{TempCand}(P,l)$ representing the candidate motion vector at block position P and hierarchical level l, based on the previously obtained motion vector $\overrightarrow{MV}_{Prev}(P,l)$ at the same block position. The top level is defined as $l = 0$. For $l = 0$, the previous motion vector is defined as the motion vector previously obtained in time at the same hierarchical level. For $l > 0$, it is defined as the motion vector previously obtained at the same frame time in the same direction, but at hierarchical level $l - 1$.

Discussion on simple motion-vector propagation

For moving objects, the uncovered background results in sub-optimal motion vectors, which do not correctly represent the motion. For bidirectional ME, a block with uncovered background can be properly predicted from either the forward or backward estimate, but not both. The vector that is not suitable is sub-optimal because it does not describe the actual motion, but merely the block most similar to the uncovered background in the direct spatial neighborhood. Propagating this vector within the same hierarchical level from forward to backward does not

provide a good vector candidate. When propagating this vector between levels (inter-level), only the forward estimation is propagated in the scheme of Figure 5.8, which is only a valid candidate for half of the cases. For this reason, the propagation scheme is modified and an enhanced motion-vector propagation scheme is proposed, which is discussed in the sequel.

Enhanced propagation of temporal vector candidates

In the previously discussed motion-vector propagation scheme, sub-optimal candidates can propagate to the next ME calculation. This propagation of sub-optimal motion-vector candidates occurs mostly at the contours of moving objects, which occasionally leads to additional coding noise. To improve on the simple propagation scheme of temporal candidates, an enhanced motion-vector propagation scheme is proposed that is adaptive to the operation-mode decisions of the coding system.

This enhanced propagation scheme creates a single motion-vector field which describes the motion more accurately for a bidirectional pair. During temporal encoding, mode decisions are made for each block, describing whether the lowest prediction residual can be obtained when using backward, bidirectional or forward prediction for motion estimation. When using enhanced propagation in HPPS, the mode decision information is re-utilized for selecting the best temporal motion-vector candidate, as shown in the following,

$$\overrightarrow{MV}_{BidirTempCand} = \begin{cases} \overrightarrow{MV}_{PrevFwd} & \text{for forward mode,} \\ (\overrightarrow{MV}_{PrevFwd} - \overrightarrow{MV}_{PrevBwd})/2 & \text{for bidirectional mode,} \\ -\overrightarrow{MV}_{PrevBwd} & \text{for backward mode,} \end{cases} \tag{5.9}$$

with $\overrightarrow{MV}_{BidirTempCand}$ denoting the resulting bidirectional candidate, $\overrightarrow{MV}_{PrevFwd}$ the previously obtained forward motion vector and $\overrightarrow{MV}_{PrevBwd}$ the previously obtained backward motion vector. All $\overrightarrow{MV}$ notations in the above and the equation are given in short-hand notation without the block position to facilitate easy formatting and reading. From Equation (5.9), it can be observed that the bidirectional candidate vector is always mapped to the forward direction, ensuring the desired symmetry.

This mode-adaptive candidate vector is then propagated in the SVC as shown in Figure 5.9, where the dashed arrows represent the propagation and the numbers above these arrows a multiplication factor. The dotted circles indicate groups of (bi)directional motion vectors. Each group only generates one candidate set and uses one set as the input. Internally, the received candidates are directly used for

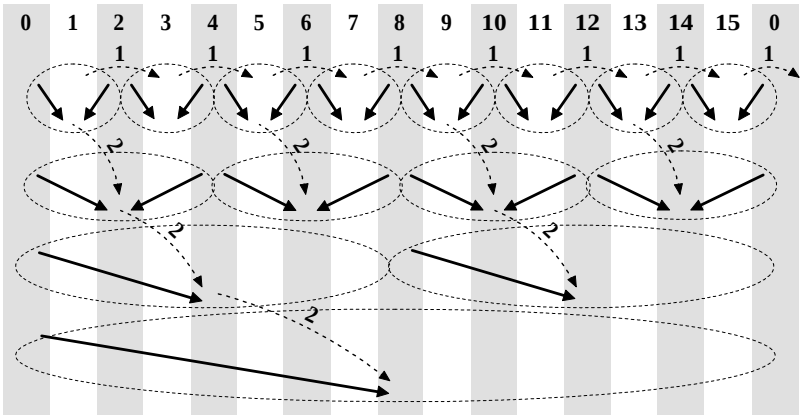


Figure 5.9: Enhanced temporal candidate propagation in a four-level SVC with every vertical stripe (columns) representing a frame, where the top numbers indicating the frame index in the GOP. Normal arrows indicate ME calculations and circles group (bi)directional vectors. Temporal candidates are represented by the dashed arrows labeled with the multiplication factor of the motion-vector field.

the forward motion estimation and inverted to be used in the backward motion estimation.

The temporal candidate prediction in Figure 5.9 can be split in two categories: intra-level (within the same hierarchical level) propagation and inter-level propagation. The temporal intra-level and inter-level motion-vector propagation $\overrightarrow{MV}$ for enhanced candidates are defined through Equations (5.10) and (5.11), respectively, giving

$$\overrightarrow{MV}_{BidirTempCand}(P,0) = \overrightarrow{MV}_{BidirPrev}(P,0), \tag{5.10}$$

$$\overrightarrow{MV}_{BidirTempCand}(P,l) = 2 * \overrightarrow{MV}_{BiDirPrev}(P,l-1) \qquad \text{for } l > 0, \tag{5.11}$$

with $\overrightarrow{MV}_{BidirTempCand}(P,l)$ representing the bidirectional motion-vector candidate at block position P and hierarchical level l, based on the previously obtained bidirectional motion vector $\overrightarrow{MV}_{BidirPrev}(P,l)$. From these equations, it can be derived that the adaptive MV propagation is similar to the non-adaptive propagation, except for the motion vector inversion. This inversion is not performed anymore because the bidirectional candidates are always mapped to the forward direction.

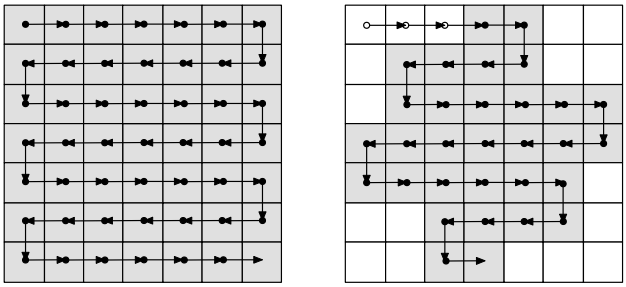
Parallelogram-Shaped Scanning (PSS) pattern

As discussed in the previous two sections, the spatial and propagated temporal candidates are used to create a spatial and temporal predictor. In HPPS, a dense search is performed around these two predictors, in a Parallelogram-Shaped Scanning (PSS) pattern.

It is proposed to use two specific processing blocks that can be applied to full-image search regions, to facilitate full exploitation of the parallel processing power of modern computing architectures. The first specific block performs SAD calculations on image blocks and the second specific unit performs a cyclic shift of image blocks. With these two basic processing kernels, it is possible to efficiently perform region searching in HPPS. Some existing computing architectures already have these processing blocks implemented as dedicated hardware, while for other processors with parallel data paths and computing cores (SIMD), these two blocks can be implemented very efficiently.

To illustrate the usage of these two specific processing blocks, a full search is performed as follows. The method is based on the principle to keep the both the predicted block and a section of the reference image at a fixed position in fast memory, and then moving the reference image below the predicted block over the search positions. By storing the reference image for a width of the block size and the search region, data can be retrieved for the next whole line at once, very effectively utilizing the wide memory load paths. During this load, single cycle SAD calculations can be performed within the line. The full-search block estimation example uses the following parameters. The block size is indicated by BS and the search range by SR, with e.g. $SR = 7$, indicating a 7×7 search area in which motion vectors are tested with a range from -3 to +3 for both horizontal and vertical positions. Two local buffers are employed, one of size $BS \times BS$ for a block from the predicted image and one of size $(BS + (SR - 1)) \times (BS + 1)$ for a block from the reference image. This reflects the actual block size plus all new vector positions of the horizontal range, plus one additional line to facilitate parallel data loading. To initialize the search, a block of size $BS \times BS$ is loaded from the predicted image and of size $(BS + (SR - 1)) \times BS$ from the reference image. In the background, the next line of size $(BS + (SR - 1)) \times 1$ is fetched from the reference image. An SAD calculation is performed on a block of size $BS \times BS$ between the predicted and the reference blocks, after which the reference image block is shifted to the left by one pixel. This shift is circular to indicate that data falling out of the window at the left is inserted at the right. The SAD calculation and circular shifting of the reference block are performed SR times, completing a single row of full SAD calculations for motion vectors with varying horizontal positions. For the last motion-vector posi-

tion in the row, instead of the circular shift to the left, the reference block is shifted one line upwards (non-circular). This up-shift inserts the previously fetched line (which was fetched in the background) in the search region. After this shift, the additional line at the bottom of the reference data block can be used to fetch a new line from the reference image in the background. For the next row of motion-vector candidates, instead of an SAD calculation and a left circular shift of the reference data, an SAD calculation is performed, followed by a right shift of the reference data. This process repeats itself for the remaining lines from the reference image. This is visualized in Figure 5.10(a) where the SAD calculations and data shifts of the reference data block are indicated for a 7×7 full-search window.



(a) (b)

Figure 5.10: Scanning patterns of HPPS for a 7×7 search window: (a) regular full search and (b) PSS pattern. The arrows indicate the effective movement of the SAD window, implemented by a circular shift of the image block in the opposite direction. Each block represents one motion-vector position and grey blocks refer to applying the SAD calculation for a particular motion vector.

To enhance the efficiency of motion estimation, the Parallelogram-Shaped Scanning (PSS) pattern is proposed, which is visualized in Figure 5.10(b). This scanning pattern utilizes the same left, right and down shifts of the reference block and the same background line fetching. However, less SAD calculations are performed, by omitting search points that are less likely to occur, which is accomplished by a modification of the shifting schedule. Table 5.1 compares the difference in SAD calculations between the full search and the PSS pattern for various search ranges. From this table, it can be seen that for larger search ranges, the number of SAD calculations for the PSS pattern converges to 50% of that of the full-search pattern. The experiments show that the PSS pattern has a negligible decrease of performance compared to the full-search pattern (typically in the order of 0.01 dB).

A possible further optimization is that the shape of the PSS pattern and its en-

Table 5.1: Comparison of SAD calculations between Full-Search (FS) and the PSS pattern for various search areas and approximately the same performance. The search range $n \times n$ represents motion vectors between $\lceil -n/2 + 0.5 \rceil$ and $\lfloor +n/2 \rfloor$ in both directions.

Search area	2×2	3×3	4×4	5×5	6×6	7×7	8×8	9×9
SAD FS	4	9	16	25	36	49	64	81
SAD PSS	4	7	12	17	24	31	36	45
Savings	0%	22%	25%	32%	33%	37%	44%	44%

Search area	10×10	11×11	12×12	13×13	14×14	15×15	16×16
SAD FS	100	121	144	169	196	225	256
SAD PSS	54	59	72	89	106	113	139
Savings	46%	51%	50%	47%	46%	50%	46%

closed number of vector positions can be adjusted in such a way that it features full utilization of the available computational power by adapting the width for the top and bottom rows of the shape. In this way, the calculation time matches the memory fetch time. For the center rows, more calculations are performed than would be optimal for the memory bandwidth, but they are required for accurate motion estimation. Through this flexible selection of motion-vector candidate positions, the memory bandwidth is fully exploited without sacrificing motion estimation performance.

During experiments, it was found that a dense search for the $(0,0)$ path only marginally improves the coding performance and can therefore be omitted to reduce the computational complexity and bandwidth usage. This can be explained by considering that the $(0,0)$ path provides a way for the ME to reset itself in case of appearing static background. In such a case, performing a dense search for these static regions would provide little or no benefit over only checking the $(0,0)$ vector.

SAD cost biasing

The fast HPPS ME algorithm is not only attractive for SVC coding, but is also applicable to surveillance object analysis and tracking. Therefore, an additional function called cost biasing is employed, to control the preference of using particular vector candidates. With this cost biasing, the motion-vector field consistency can be influenced, which may be partially beneficial for e.g. surveillance object analysis and tracking.

The HPPS ME algorithm provides a function to bias the SAD cost of the candidate motion vectors, thereby effectively making certain motion-vector candidates less likely to be selected. Cost biasing is performed according to Equation (5.12), which specifies

$$CostBias(dx, dy) = \alpha \cdot (abs(dx) + abs(dy)) + \beta, \tag{5.12}$$

with $CostBias(dx, dy)$ being the SAD cost bias for the tested motion vector represented by the displacement (dx, dy), with α and β the parameters for HPPS cost biasing. The biased SAD can now be derived from Equation (5.1) with the shorthand notation of the SAD and is specified by

$$SAD_{Biased}(x, y, dx, dy) = SAD(x, y, dx, dy) + CostBias(dx, dy), \tag{5.13}$$

with $SAD(x, y, dx, dy)$ being the SAD of a block at pixel location (x, y) with the tested motion vector represented by the displacement (dx, dy).

Cost biasing can be applied to create a more smooth motion-vector field through emphasizing the importance of the exact spatial and temporal candidates by increasing the α parameter. The β can be used to bias the motion estimation to the $(0, 0)$ vector to avoid noisy vectors selection, which can occur for static objects due to image noise. This parameter can also be applied for biasing towards spatial or temporal vector selection consistency.

From the experiments, it was found that for our purpose of video compression, the smoothing of motion-vector selection via cost biasing reduces the coding performance and is thus not desirable, so that the α parameter can be set to $\alpha = 0$. For the coding of surveillance scenes, there is a preference for the $(0, 0)$ motion vector, since it can be more efficiently encoded and background parts are typically static. This preference can be realized by setting $\beta > 0$ for the spatial and temporal candidate paths. The aspect of utilizing motion vectors for content analysis is described in more detail by Vijverberg *et al.* in [108].

For the SVC coding case, it has already been mentioned the cost biasing is optional. It was discussed in the previous section that a dense search around the $(0, 0)$ vector provides little or no benefits. Without a search region, the α parameter becomes useless. Furthermore, with the preference of the $(0, 0)$ vector, the β parameter is set to $\beta = 0$.

5.4 Experimental Results

In this section, the proposed HPPS ME algorithm is evaluated. The same test sequences as used in Chapter 4 are employed, of which an overview is given in

Appendix E. Similar to Chapter 4, intermediate results regarding sub-parts are presented for a subset of the sequences, in this case the well-known City sequence. At the end of this section, the final results for all sequences are presented.

To evaluate the effectiveness of the HPPS ME algorithm, two experiments are conducted. The first involves measuring the quality of the motion-compensated frames compared to the original frames. This enables for a pure comparison of the quality of the ME algorithm without influence of any spatial and temporal coding techniques. The temporal structure of the 4-level BDLD configuration (Figure 5.2) is utilized. This leads to a GOP size of 16 and motion to be estimated for frames 1, 2, 4 and 8 frames apart. This analysis also allows for measuring changes at various levels in the temporal hierarchical tree. The second experiment estimates the impact of the ME algorithm on the full coding process, which is presented under the condition of utilizing the full proposed SVC codec and the two best performing ME algorithms. The full-search ME algorithm is used as a reference throughout this section. The ME parameters utilized for evaluation are listed in Table 5.2.

Table 5.2: ME parameters used during evaluation with SR = Search Range. Non-listed parameters are set to their default values as given in Section 5.2.

Context	Parameter	Setting
All	Blocksize	16×16 pixels
Measurements	Sub-pixel estimation	None
Full codec	Sub-pixel estimation	Quarter
HPPS	Cost biasing	Equal SAD: prefer $(0,0)$ vector
HPPS	Search Range (SR)	± 4 pixels for all levels
EPZS	Maximum iterations	16
ARPS-3 & 3DRS	Allowed vectors	Limited to FS search ranges
Full-search	SR at 1, 2, 4 and 8 frames apart	± 8, ± 16, ± 32, ± 32 pixels

This section is structured as follows. Section 5.4 verifies the performance of HPPS against other well-known and state-of-the-art ME algorithms, while focusing on the hierarchical temporal structure commonly used in SVC. Section 5.4 investigates the enhanced candidates for both the state-of-the-art EPZS and the proposed HPPS. Finally, Section 5.4 evaluates the impact of EPZS and HPPS on the rate-distortion performance of the complete SVC implementation, with and without enhanced candidates.

Verification against other ME algorithms

The performance of the proposed HPPS ME algorithm has been compared with other well-known algorithms: Full Search, ARPS-3, 3DRS and EPZS. The performance is measured by calculating the PSNR of the motion-compensated image compared to the original image. Since the interest is on estimating the performance of ME in SVCs, this PSNR measurement is performed for each of the four levels in the BDLD configuration. Figures 5.11 and 5.12 show the results of these measurements for the well-known City sequence. Furthermore, the number of SAD calculations have been measured and these numbers are depicted in Table 5.3.

Table 5.3: Average and maximum SAD counts per block for various ME algorithms at different levels in the temporal hierarchical tree, averaged over 160 frames of the City sequence.

	3DRS SADs		ARPS-3 SADs		EPZS SADs		HPPS SADs		FS SADs	
Level	mean	max	mean	max	mean	max	mean	max	mean	max
1	4	9	8	28	9	35	86	91	275	289
2	5	9	10	47	10	25	86	91	1036	1089
3	5	9	11	52	11	37	86	91	3916	4225
4	4	9	14	54	12	40	86	91	3916	4225

In Figure 5.11(a) and the first row of Table 5.3, a comparison for the lowest level in the SVC is shown with a temporal distance of 1 frame and featuring bidirectional motion estimation. It can be observed that FS and HPPS show nearly identical performance. EPZS and ARPS-3 perform a bit worse for parts of the sequence and 3DRS has a lower performance, but was not designed for video coding applications and has a considerably lower complexity in terms of SAD.

For EPZS and ARPS-3, the amount of SAD vector calculations is clearly lower than HPPS and FS, which explains the somewhat lower performance. The average amount of SAD calculations is in the order of 10 vector estimations, so that in normal conditions they perform quite well. However, as soon as special signal or motion transients occur, the algorithms will require much more computations and will grow to the order of 50 vector candidates. The HPPS algorithm is memory-bandwidth driven and computes a larger but fixed amount of vectors, and should be compared to full search at this level of operation. In this case, HPPS is a factor of 3.2$\times$ more efficient in terms of SAD calculations than full search, while offering nearly the same performance. Furthermore, HPPS always retrieves two times 24 $\times$ 24 pixels for estimating one predicted block, versus 32 $\times$ 32 pixels for FS at this level. The bandwidth for ARPS-3 and EPZS can hardly be estimated, due to the iterative process of converging to the best vector.

Figure 5.11(b) shows the next level of bidirectional motion estimation, with frame distances of two. The performance of the various ME algorithms is similar to the previously discussed level. However, the average level of performance is somewhat decreased due to the higher temporal distance, and curves start to deviate from each other slightly.

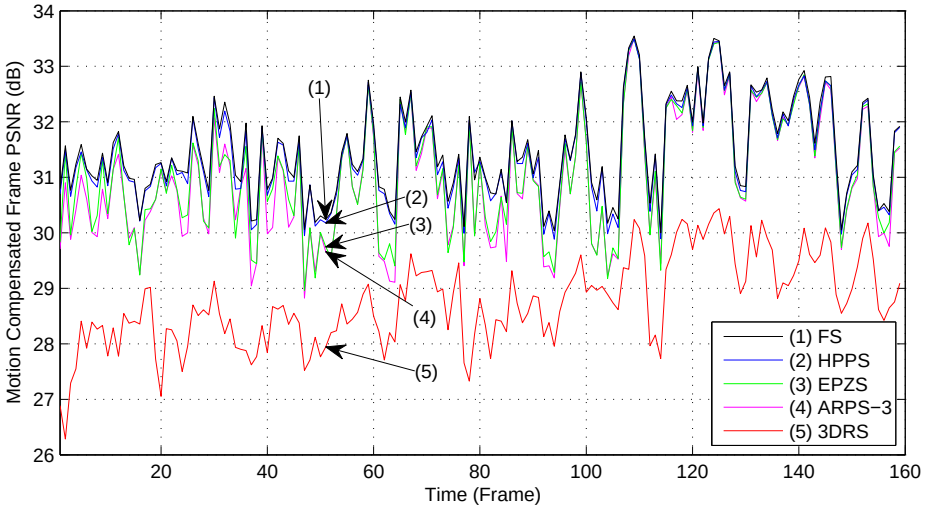
The performance results for the next two levels, with frame distances of 4 and 8, are visualized in Figure 5.12(a) and Figure 5.12(b), respectively. At these frame distances, the ARPS-3 also starts to lose accuracy, while HPPS performs a bit above EPZS. At this level of temporal processing, EPZS is still surprisingly efficient with respect to SAD calculations, but requires slightly more iterations to evolve to the best vector. For EPZS, the amount and path of iterations is undefined and this also holds for the cache access to the video data. In case of the HPPS algorithm, the results are obtained with a fixed and limited data-access pattern, which can be very effectively cached (e.g. pre-fetching using DMA). Again HPPS is best compared with full search, where HPPS is 46× more efficient with respect to the amount of SAD calculations and requires 5.6× less bandwidth at these levels.

Simple and enhanced temporal candidates

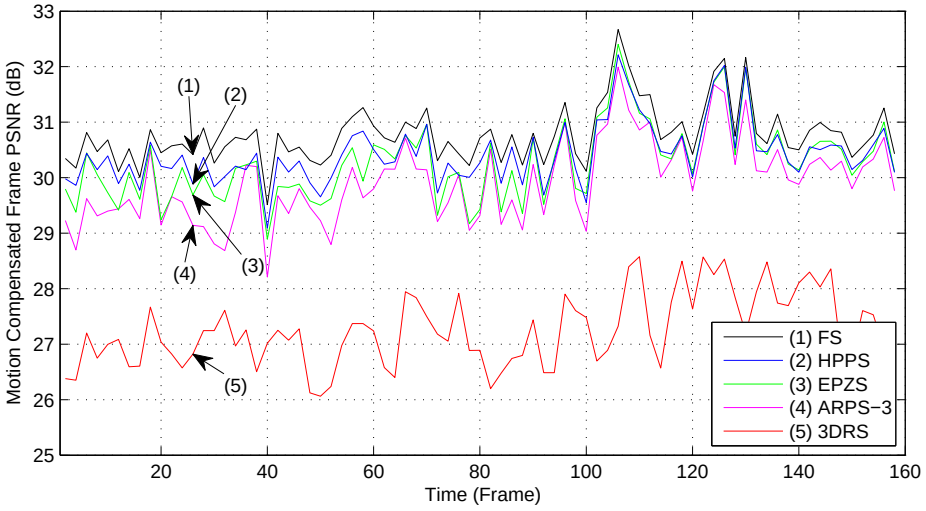
This subsection examines the effectiveness of the proposed ME algorithm with simple and enhanced candidates. The evaluation involves PSNR measurements at the 4 levels of the BDLD configuration with both the HPPS and EPZS ME algorithms. Figures 5.13 and 5.14 show the results of these measurements for the well-known City sequence, where the PSNR is calculated by comparing the original image with the motion-compensated image.

From Figure 5.13(a), it can be seen that the improvement for the lower levels in HPPS is only marginal. However, at higher levels, Figures 5.13(b), 5.14(a) and 5.14(b) show that the improvement is significant. This can also be observed from Table 5.4, where the mean PSNR for the various configurations is listed. There is no improvement at Level 1 (Δframes=1), but at Level 3 (Δframes=4) the improvement is most pronounced at 1.14 dB. It can be noted that the improvement at Level 4 (Δframes=8) is lower, however, no bidirectional estimation exists at Level 3, so this improvement is fully due to the improved motion vectors at Level 3.

For EPZS, only a very limited improvement can be observed at higher levels from both Figures 5.13 and 5.14 and Table 5.4 (bottom), most likely due to the fact that EPZS always evaluates four temporal predictors, thereby already producing a more stable result than a system with only one temporal predictor, such as HPPS.

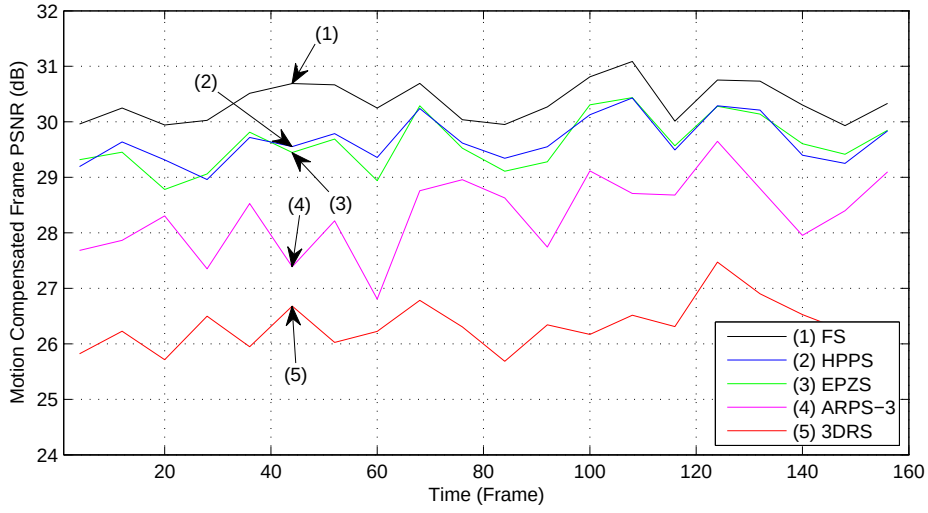


(a)

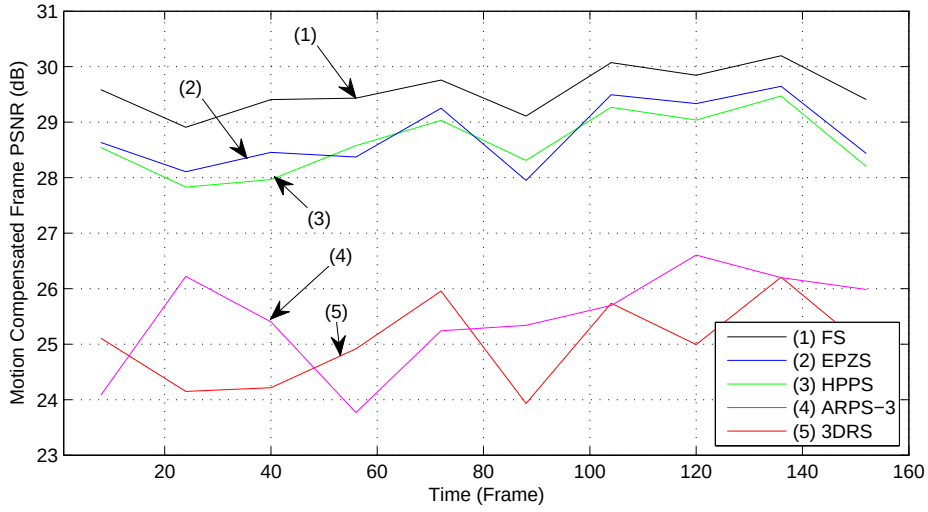


(b)

Figure 5.11: PSNR measurements per frame for the City sequence using the Full-Search, HPPS, EPZS, ARPS-3 and 3DRS ME algorithms for temporal distances of: (a) 1 and (b) 2 frames apart. Curves are sorted from 1 to 5 by performance in mean PSNR.

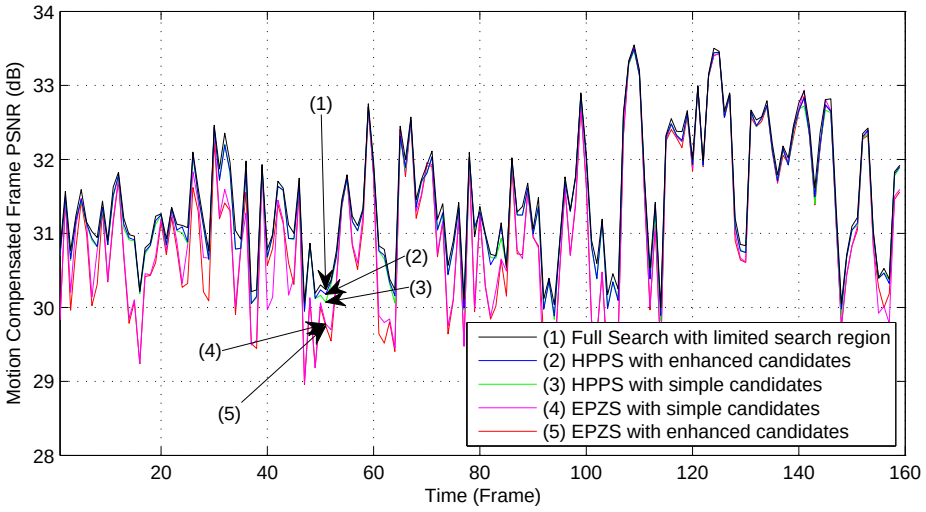


(a)

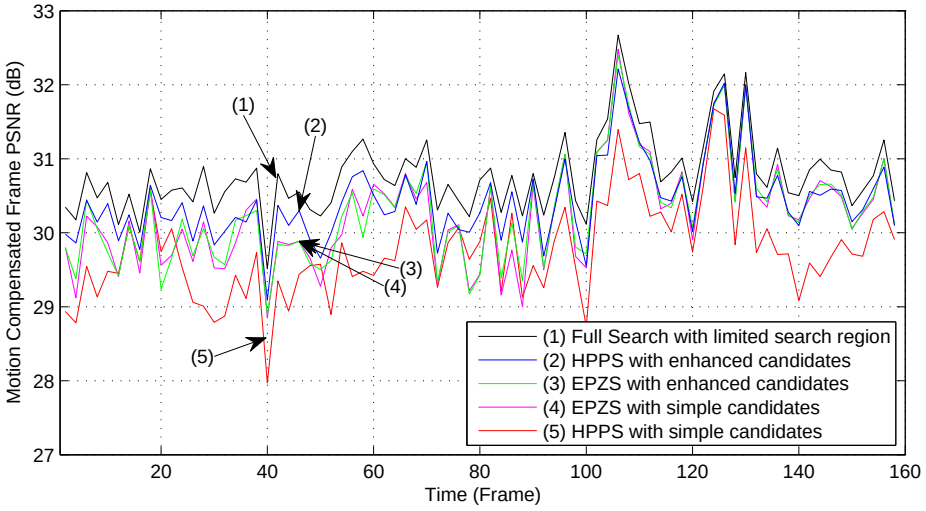


(b)

Figure 5.12: PSNR measurements per frame for the City sequence using the Full-Search, HPPS, EPZS, ARPS-3 and 3DRS ME algorithms for temporal distances of: (a) 4 and (b) 8 frames apart. Curves are sorted from 1 to 5 by performance in mean PSNR.

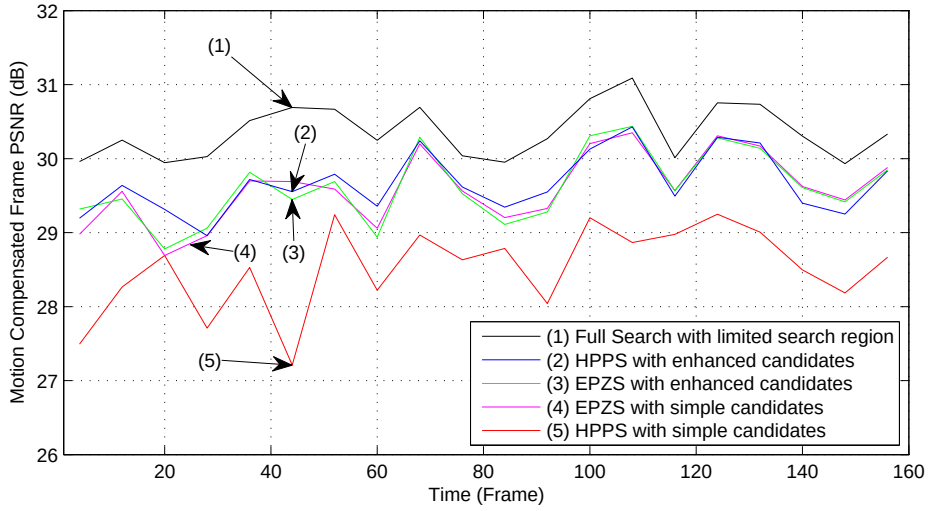


(a)

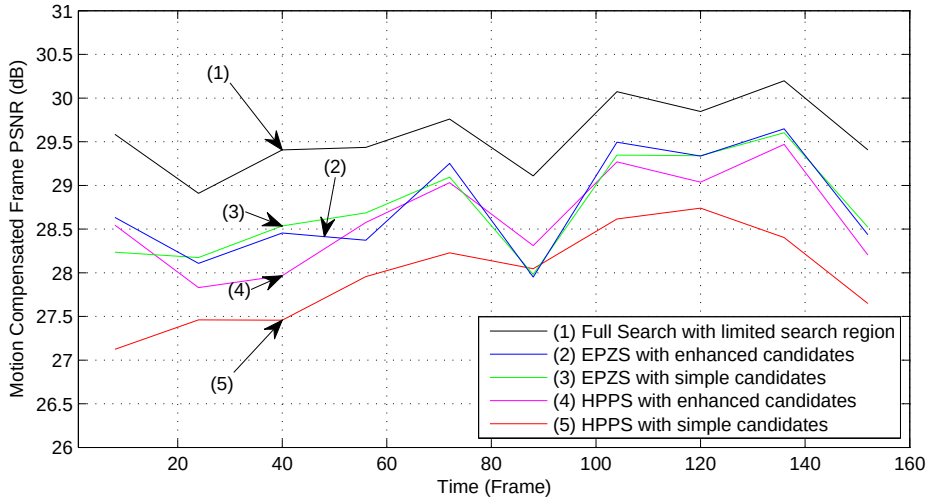


(b)

Figure 5.13: PSNR measurements per frame for the City sequence using the Full-Search, HPPS simple and enhanced and EPZS simple and enhanced ME algorithms for temporal distances of: (a) 1 and (b) 2 frames apart. Curves are sorted from 1 to 5 by performance in mean PSNR.



(a)



(b)

Figure 5.14: PSNR measurements per frame for the City sequence using the Full-Search, HPPS simple and enhanced and EPZS simple and enhanced ME algorithms for temporal distances of: (a) 4 and (b) 8 frames apart. Curves are sorted from 1 to 5 by performance in mean PSNR.

Table 5.4: Mean PSNR of the motion-compensated frame at various temporal levels for HPPS (top) and EPZS (bottom), with simple and enhanced candidates.

Level (Δframes)	1 (1)	2 (2)	3 (4)	4 (8)
HPPS simple	31.39 dB	29.77 dB	28.52 dB	27.97 dB
HPPS enhanced	31.40 dB	30.43 dB	29.67 dB	28.62 dB
Improvement	0.01 dB	0.66 dB	1.14 dB	0.66 dB

Level (Δframes)	1 (1)	2 (2)	3 (4)	4 (8)
EPZS simple	31.11 dB	30.25 dB	29.60 dB	28.75 dB
EPZS enhanced	31.07 dB	30.28 dB	29.62 dB	28.77 dB
Improvement	-0.04 dB	0.03 dB	0.02 dB	0.02 dB

Impact on Rate Distortion (RD) in the complete SVC

In this section, the performances of the two best performing ME algorithms (HPPS and EPZS) are measured, paired with both simple and enhanced candidates. To this end, these two ME algorithms are implemented in our complete SVC framework. The codec is set up as follows. The spatial wavelet transform utilizes the 5/3 integer wavelet transform, based on lifting (Section 2.3) with a 5-level dyadic structure (Section 2.3) and 2D energy correction (Section 2.3). The temporal decomposition employs the proposed 4-level BDLD configuration (Section 4.3) with a GOP size of 16 frames. Entropy coding is performed using the proposed TSSP codec (Section 3.3) with both extensions: 5/3 energy-correction mode (Section 3.4) and highly scalable mode (Section 3.4). Sequences are encoded once at the highest quality to a single encoded bitstream using the TSSP encoder. Lower-quality bitstreams are then extracted from this bitstream using the TSSP parser, after which they are decoded using the TSSP decoder. The coding system is equipped with the proposed extensions to the typical t+2D structure: temporal energy correction (Section 4.4) and low-complexity encoder feedback (Section 4.4), where the quality-reduced frame is based on a BPR of 3.

Figure 5.15 shows the rate-distortion curve for the City sequence using FS, HPPS and EPZS ME algorithms, with simple and enhanced candidates. In Figure 5.15(a), the complete rate-distortion curve is presented, of which a region around 3 Mbit/s is enlarged in Figure 5.15(b). It can be clearly observed from these figures that, when implemented in the full coding system, both HPPS and EPZS provide very robust ME and approach the FS algorithm within 0.1 dB.

Regarding the simple and enhanced candidate propagation, the enlarged view

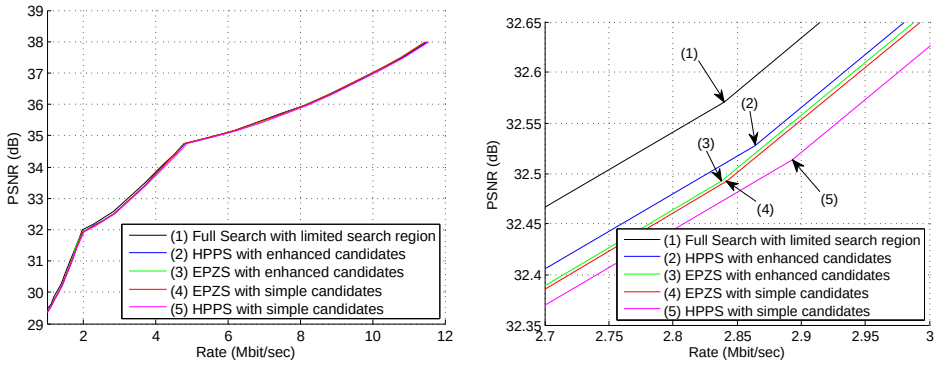


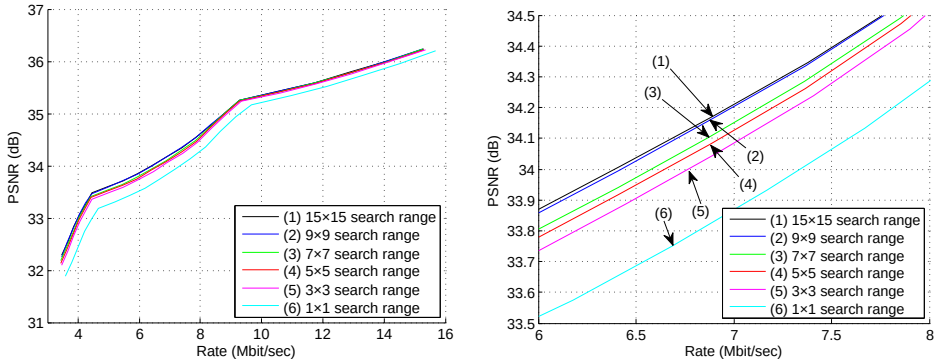
Figure 5.15: Rate-Distortion (RD) curve for the City sequence at 30 fps using the Full-Search, HPPS with simple and enhanced candidates and EPZS with simple and enhanced candidates, with (a) overview and (b) zoom at 3 Mbit/s. Curves are sorted from 1 to 5 by performance in RD.

of Figure 5.15(b) shows that the enhanced candidates only provide a small gain for EPZS and a larger gain for HPPS, similar to the results found in the previous section. With the enhanced predictors, HPPS now even slightly outperforms EPZS in this particular case.

Figures 5.18 and 5.19 provide the rate-distortion results for all sequences of Appendix E using the reference FS, the state-of-the-art EPZS and the proposed HPPS ME algorithms with enhanced candidates. From these figures, it can be concluded that the EPZS and HPPS ME algorithms have a nearly identical performance as the reference FS ME algorithm for all sequences. The difference between the algorithms is within 0.05 dB for SD sequences and 0.03 dB for HD sequences. These differences are negligible and both algorithms are very good candidates for fast motion estimation. However, it should be noted that EPZS has a serial processing nature and requires a variable processing time, while the HPPS is optimized for *parallel* implementations and has a *fixed* processing time.

The above results for HPPS are obtained using a Search Range (SR) of ± 7 pixels for all levels. To reduce computational complexity, the SR can be reduced, which will result in a lower RD performance. The RD curves for the City sequence with varying SR are presented in Figure 5.16. From this figure it can be observed that the performance with SR=± 7 pixels (Curve 1) is nearly identical to the performance with SR=± 4 pixels (Curve 2). For lower SRs, the performance reduces gradually.

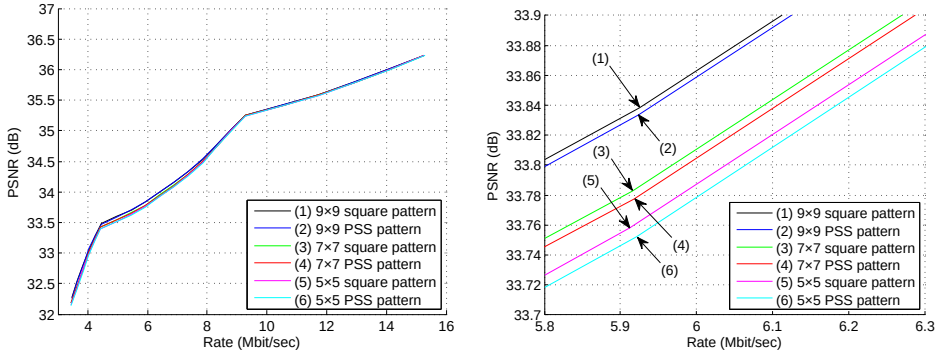
For all experiments above, the PSS pattern has been used within the HPPS ME



(a) (b)

Figure 5.16: Rate-Distortion (RD) curve for the City sequence at 60 fps using HPPS with enhanced candidates for various search ranges, with (a) overview and (b) zoom at 7 Mbit/s. Curves are sorted from 1 to 6 by performance in RD.

algorithm. To give an indication of the effectiveness of the PSS scanning pattern, Figure 5.17 shows the RD curves for the City sequence, in which the PSS pattern is compared with the square pattern for varying search ranges. From this figure, it can be observed that the performance of HPPS employing the PSS pattern is within 0.01 dB of HPPS using the square pattern.



(a) (b)

Figure 5.17: Rate-Distortion (RD) curve for the City sequence at 60 fps using HPPS with enhanced candidates, for square and PSS search patterns and various search ranges, with (a) overview and (b) zoom at 6 Mbit/s. Curves are sorted from 1 to 6 by performance in RD.

5.5 Conclusions

This chapter has presented Motion Estimation (ME) from the perspective of using such processing in a Scalable Video Coding (SVC) system and implementing this on modern multimedia computing architectures. The combination of both views require a reconsideration of the design criteria for ME algorithms. SVC provides the ME algorithm with frames at varying temporal distances, where new methods of motion-vector propagation are necessary. Hardware accelerators in modern computing architectures shift the complexity bottleneck from SAD computations to the required memory bandwidth. Both aspects have motivated the choice for a new design of the ME algorithm specifically for the SVC framework and its mapping on a modern multimedia processor.

The first part of this chapter evaluates several well-known ME algorithms: ARPS-3, PMVFAST, EPZS and 3DRS. These ME algorithms are all based on the regular assumptions that the SAD calculation is the computational bottleneck and motion is estimated in the majority for sequential frames and/or with limited temporal distances. For our application, the novel proposed ME algorithm is based on the following design aspects: (1) ME suited for SVC and its support for bidirectional and temporal coding hierarchies and larger temporal distances (e.g. 8 frames), (2) ME algorithm matches with block-shifting accelerators and block-SAD calculations as found in modern media computing architectures and (3) emphasis on regular data-access patterns and lower priority on the amount of SAD calculations.

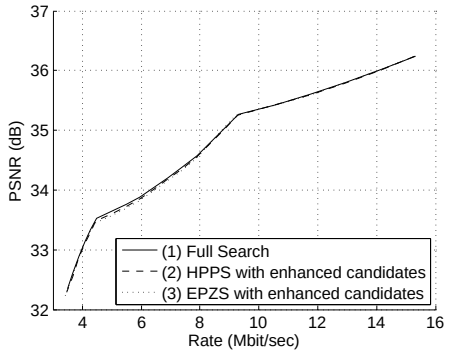
The novel Highly Parallel Predictive Search (HPPS) ME algorithm differs from previous proposals in that it utilizes three independent processing paths: one for the $(0,0)$ vector, one using a spatial predictor and one for a temporal predictor. The two predictive candidate vectors are surrounded with additional refinement candidates arranged in a dense PSS pattern, which reduces SAD test points up to 50% without significantly reducing the accuracy of the found motion vectors. This robust combination of a predictive candidate with a dense search allows a straightforward motion-vector candidate generation for an SVC system. Moreover, the size of the PSS pattern around both candidates can be reasonably small, but large motion vectors can still be found due to the hierarchical, multi-level candidate propagation. Furthermore, both the SAD calculation and the scanning of the PSS pattern can be parallelized and mapped to multi-core architectures in a highly efficient manner. The computational load of HPPS is fixed, regardless of scene activity and temporal distance, which guarantees a fixed computational load, considered essential for real-time systems.

HPPS performs well for various levels in an SVC and shows a similar perfor-

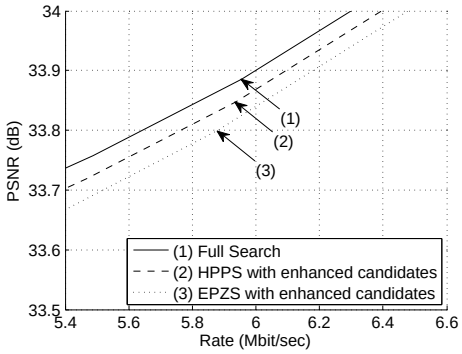
mance as the state-of-the-art EPZS ME algorithm. Enhanced candidates within the SVC hierarchical tree improve HPPS performance by up to 1.14 dB with respect to the quality of the motion-compensated frame. With enhanced candidates, both the HPPS and EPZS ME algorithm approach the full-search reference within 0.05 dB for SD sequences and 0.03 dB for HD sequences.

The performance of EPZS is considered state-of-the-art and HPPS has a similar performance with respect to the measured rate-distortion curve. The main benefit of HPPS lies in its characteristic features with respect to real-time implementation. First, HPPS has a fixed computational load, regardless of scene activity and the temporal distance between frames. Second, HPPS is very predictable in its data usage, as only three predictors are explored (with one used for the (0,0) vector), with a pre-determined search region around each predictor. For comparison, EPZS uses already 4 predictors for temporal processing. HPPS tests more motion-vector locations, but the pattern has a high data-locality so that cache data-access is minimized. For comparison ARPS-3 and EPZS have a lower cost in computation, but require a variable amount of data-access operations as the algorithms are iterative.

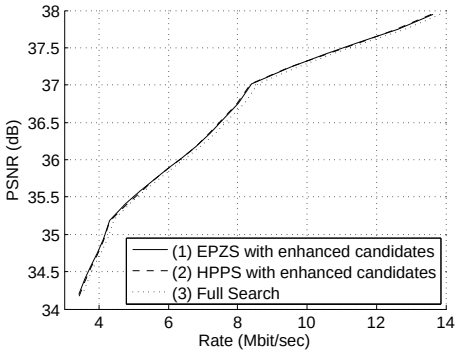
At this point, all relevant stages of the proposed scalable video coding system have been discussed, including its various scalability features, low complexity and suitability for embedded implementation. In Chapter 6, the implementation aspects of the considered SVC will be validated by implementing the proposed coding system on a Digital Signal Processor (DSP) within a surveillance camera.



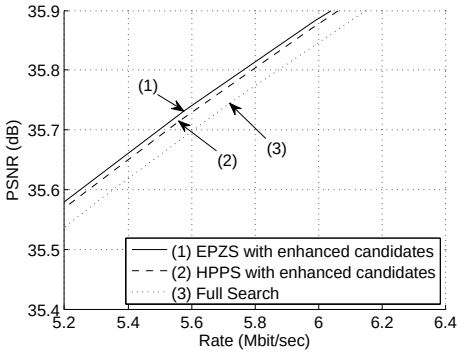
(a) City 704 × 576 pixels, 60 fps



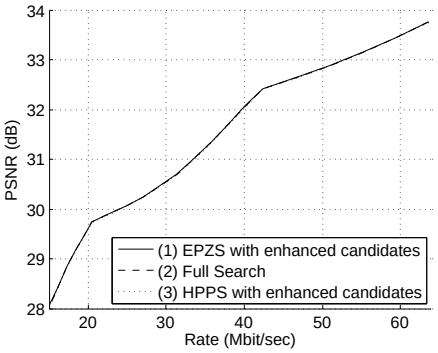
(b) (Detail) City 704 × 576 pixels, 60 fps



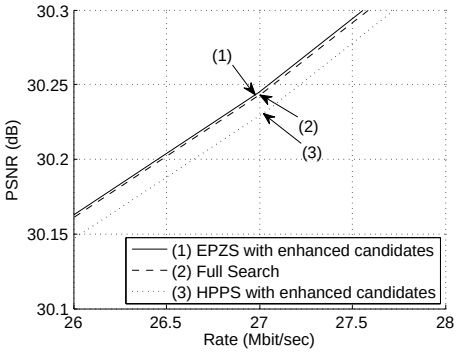
(c) Crew 704 × 576 pixels, 60 fps



(d) (Detail) Crew 704 × 576 pixels, 60 fps

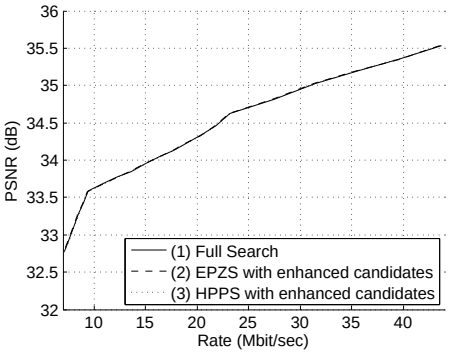


(e) Park run 1280 × 720 pixels, 50 fps

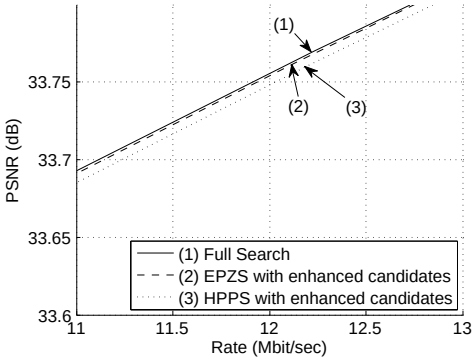


(f) (Detail) Park run 1280 × 720 pixels, 50 fps

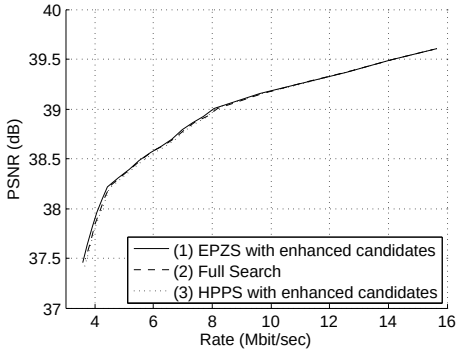
Figure 5.18: Final results for the proposed coding system, using the BDLD temporal configuration and the top ME algorithms for: (a) City, (b) City detail, (c) Crew, (d) Crew detail, (e) Park Run, (f) Park Run detail. Curves are sorted from 1 to 3 by performance in RD.



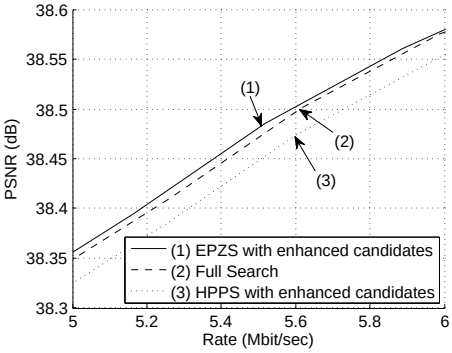
(a) Stockholm 1280 × 720 pixels, 60 fps



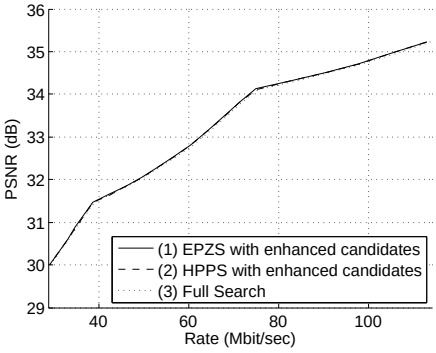
(b) (Detail) Stockholm 1280 × 720 pixels, 60 fps



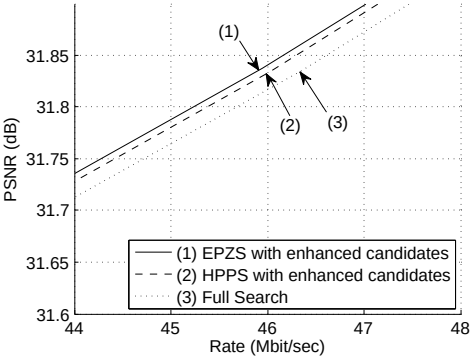
(c) Station 2 1920 × 1080 pixels, 25 fps



(d) (Detail) Station 2 1920 × 1080 pixels, 25 fps

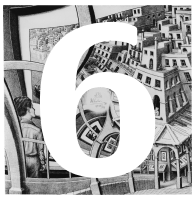


(e) Crowd Run 1920 × 1080 pixels, 50 fps



(f) (Detail) Crowd Run 1920 × 1080 pixels, 50 fps

Figure 5.19: Final results for the proposed coding system, using the BDLD temporal configuration and the top ME algorithms for: (a) Stockholm, (b) Stockholm detail, (c) Station 2, (d) Station 2 detail, (e) Crowd Run, (f) Crowd Run detail. Curves are sorted from 1 to 3 by performance in RD.

CHAPTER 

Real-time algorithmic validation on an embedded architecture

There are no such things as applied sciences,
only applications of science.

LOUIS PASTEUR
(1822 - 1895)

Abstract

This chapter concentrates on the mapping of the developed coding system onto a common DSP platform suited for embedding in a surveillance camera. The presented mapping involves three important processing stages of the developed SVC framework: optimized implementations of (1) the two-dimensional wavelet transform, (2) the TSSP wavelet coefficient encoder and (3) the temporal filtering framework. The efficient implementations have been achieved through elegant use of SIMD instructions and Direct Memory Access (DMA). For the wavelet filtering, the proposed background DMA structure is so effective, that the ALUs are practically always supplied with data input, so that the execution of a 4-level transform at 4CIF (CCIR-601) broadcast resolution only takes 6.08 ms. For TSSP, the complete implementation results in a throughput rate of 75 TSSP encoding cycles per second, thereby satisfying the performance requirements for a real-time image/video encoding system. Finally, the full temporal filtering is integrated, but without motion estimation and compensation, due to architecture limitations. Fortunately, by including the temporal filtering structure, the overall efficiency of the algorithm increases due to the decorrelation of the images. As a result, the fully scalable video encoding system can be executed at 20 fps.

6.1 Introduction

Up to this point, the algorithmic description and corresponding motivation of the proposed Scalable Video Codec (SVC) have been discussed. All of the developed algorithms in the SVC have been deliberately designed with embedded-system implementation feasibility in mind, while retaining advanced scalability features. This chapter validates the efficient implementation of the SVC system for a programmable DSP-based architecture.

As discussed in Chapter 1, the requirements for the implementation of the SVC framework are that the system should be of reasonable cost and size and avoid excessive power consumption or forced cooling. As a consequence, the adopted computing platform for a feasibility study of the execution of the SVC framework is a resource-constrained embedded system. This study involves the key processing stages of the SVC framework: (1) the spatial wavelet transform, (2) the TSSP wavelet coefficient encoding algorithm and (3) the temporal filter structure.

For the mapping, we have chosen a common Digital Signal Processor (DSP) based on a Very-Long Instruction Word (VLIW) architecture with 8 parallel processing cores. Furthermore, advanced optimization techniques and specific hardware accelerators are utilized, such as Single-Instruction Multiple Data (SIMD) parallel instructions and Direct Memory Access (DMA). The SIMD instructions facilitate data and computational parallelism and DMA is employed to transfer data to and from our self-managed Level-2 cache, parallel to computations.

This chapter provides 3 contributions. The first contribution proposes a novel way of advanced cache management with DMA- and SIMD-optimized filter kernels for multi-level 2D wavelet transforms. Second, similar techniques are used to implement the TSSP encoding algorithm. Third, it is shown that the temporal coding structure is executed and implements the proposed SVC with scalable complexity and performance. The framework is executed at 4CIF broadcast resolution.

The organization and relations of this chapter are given in Figure 6.1. Section 6.2 explains the hardware and software architecture and the available optimization features found in the DSP. The implementation of the multi-level integer wavelet transform is presented in Section 6.3, while Section 6.4 describes the implementation of the TSSP wavelet coefficient encoding algorithm. Section 6.5 discusses the implementation of the temporal filtering, while the experimental results for each of the parts and the joint operation are given in Section 6.6. Finally, Section 6.7 concludes this chapter.

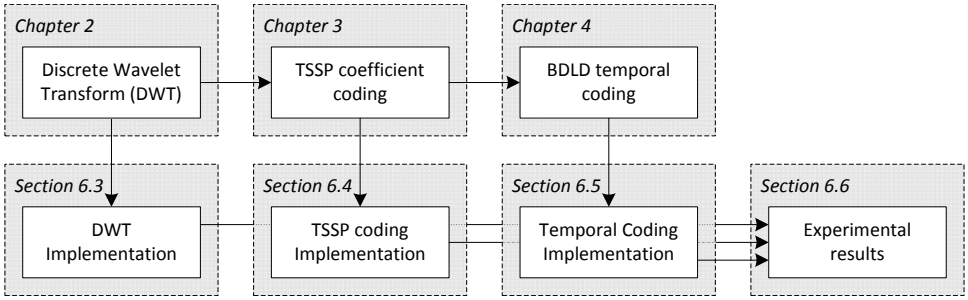


Figure 6.1: Organization of Chapter 6 and the relations to other chapters.

6.2 Video surveillance architecture used for validation

Motivation for chosen hardware platform

The system requirements from the previous section on cost, size and power consumption clearly influence the choice of the computing architecture. The following four principle categories of consideration are: (1) Application Specific Integrated Circuits (ASICs), (2) Field Programmable Gate Arrays (FPGAs), (3) generic CPUs (e.g. desktop processors) and (4) Digital Signal Processors (DSPs). The ASIC offers low power usage during execution, but requires high start up costs and lacks the desired flexibility for professional applications. The FPGA provides this flexibility, but it is a difficult device for programming complex algorithms such as video coding. If this mapping becomes inefficient because of the complexity, a larger device is required which increases cost rapidly. A generic CPU is the best choice for programming and flexibility, but consumes a significant amount of power leading to special cooling requirements. Finally, the DSP provides a balance between flexibility, ease of programming, cost and power consumption. Modern DSPs offer high computing power due to Very Long Instruction Word (VLIW) architectures with wide data paths and many options for parallel processing and efficient memory transfers. For this reason, the DSP is adopted as the preferred computing platform.

Embedding the DSP in a camera architecture

The adopted DSP computing system is embedded in a custom-designed video surveillance camera to verify the implementation of the SVC. The hardware architecture of the video surveillance camera is shown in Figure 6.2. Light coming through the lens is converted to electrical signals in the CCD sensor and digitized

for each pixel by an A/D converter. The raw video stream is then processed and compressed by the DSP and communication (control in, video out) is established through the network interface. Various input/output interfaces are defined for audio, digital I/O and auto iris control, if a lens with iris is used. The DSP is connected to fast Random Access Memory (RAM) and flash memory that stores the program and settings. A clock circuit provides timing for the DSP and other hardware components, while a watchdog timer is present to reset the DSP in case of a software malfunction or power instabilities that cause the DSP to halt. For surveillance systems, this watchdog timer is essential because systems need to be always in operation. Finally, a power supply is contained for direct supply connection, or Power Over Ethernet (POE).

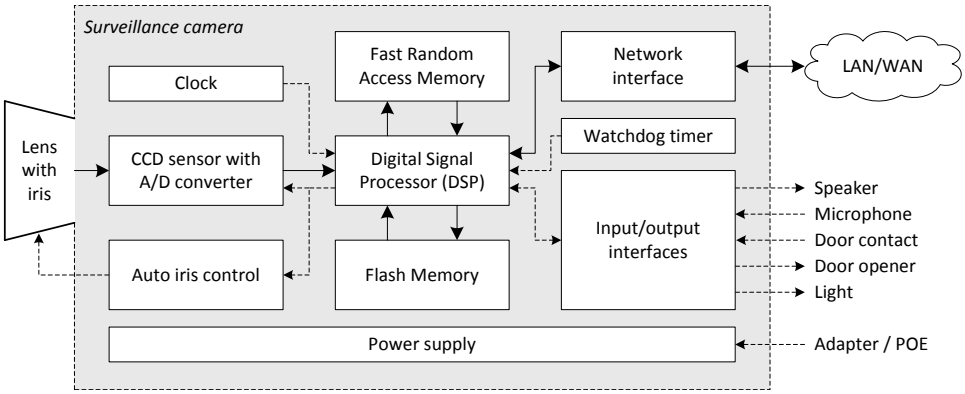


Figure 6.2: Hardware architecture of a custom commercial video surveillance camera with embedded DSP. Solid arrows indicate high-bandwidth data, dotted arrows indicate control and information data.

Besides the above-discussed video surveillance camera, also a custom video encoder was developed, which facilitates upgrading of current analog video surveillance systems to modern IP-based SVC systems. The hardware architectures of the camera and video encoder are similar, but for the video encoder, the CCD sensor is replaced by a video capturing chip and the auto iris control is removed.

DSP architecture and optimization features

The DSP architecture provides two essential features for achieving a real-time implementation: SIMD instructions (Single Instruction Multiple Data) and DMA (Direct Memory Access). SIMD allows processing of multiple data points in parallel, while executing a single instruction, thereby fully exploiting the wide data paths.

The DMA manager can transfer large amounts of data in the background, while the processor remains available for calculations.

The DSP architecture is show in Figure 6.3(a), with the VLIW computing core explained in more detail in Figure 6.3(b). For experiments, we have adopted the DM642 DSP which has been broadly accepted for many video applications. The DM642 is an 8-core VLIW processor, operating at a clock frequency of 600 MHz with an instruction width of 256 bits. Memory consists of an 133-MHz DDR (Double Data Rate) RAM connected by a 64-bit bus with three cache memories in the core: a Level-1 data cache, a Level-1 program cache and a combined Level-2 cache. The memory bandwidth of the various memories are listed in Table 6.1. Furthermore, the core has 4 load and 4 store paths of each 32 bits with special methods for 64-bit access.

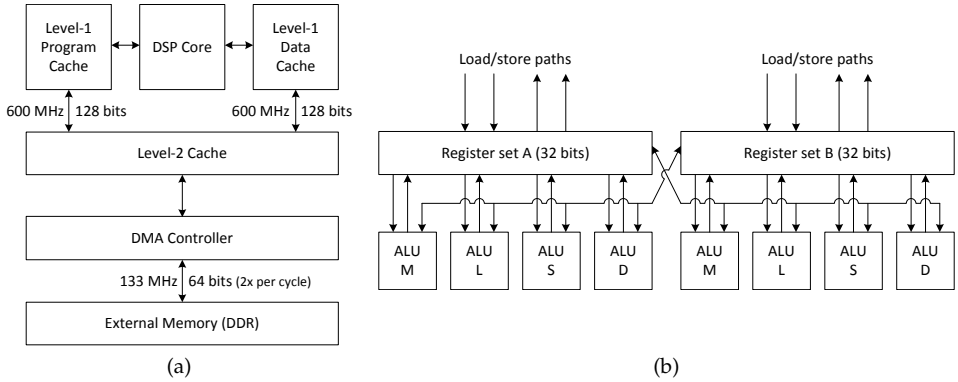


Figure 6.3: DM642 DSP (a) architecture and (b) its VLIW core with 8 parallel computing units (ALUs).

Table 6.1: Bandwidth of memory busses in the DM642 processor. Both edges of the 133 MHz clock of the external DDR memory are used for data transfer, effectively doubling the bandwidth.

Memory	Clock	Bus	Bandwidth	Description
Level-1 program	600 MHz	256 bits	19.2 GB/s	2-way set associative
Level-1 data	600 MHz	64 bits $\times 2$	9.6 GB/s	2-way set associative
Level-2 cache	600 MHz	64 bits $\times 2$	9.6 GB/s	
External (DDR)	133 MHz	64 bits	2.128 GB/s	1-way: load or store

Software architecture

Besides the custom-designed camera architecture, a dedicated software architecture is employed for the experimental video surveillance camera. This software architecture is shown in Figure 6.4. In this figure, it can be observed that the CCD sensor captures the scene information using a RGGB Bayer pattern, after which the input image is prepared for further image processing by de-Bayering the RGGB color information and converting it to the YC_bC_r color domain, while applying white balance and gamma correction. The input image is also used for exposure control, which manipulates the shutter time, CCD gain and iris aperture. After the image is converted to the YC_bC_r color space, it is compressed by the SVC. First temporal filtering is performed, then the spatial filtering and finally the entropy encoding. The compressed data is then passed to the network stack for external communication. Camera control is available through a web-based interface, so that the user can remotely adjust parameters such as resolution, image quality and other settings. Input/output drivers provide access to external equipment such as a microphone or door contacts. All the above-mentioned processing is scheduled and controlled by a basic Real-Time Operating System (RTOS).

Figure 6.4 clearly shows the important role of video coding in the video surveillance camera software architecture, but the architecture simultaneously executes a broad selection of other processing and control functions. The remainder of this chapter will focus solely on the video coding stages within the software architecture, which are enclosed by the grey block in the figure.

Regarding the video encoder, pre-processing is significantly less complex, because the following typical pre-processing steps are absent in the software architecture: de-Bayering, gamma correction, white balance and exposure control. Despite the absence of those functions, the video coding stages are unchanged.

Development platform and profiling environment

The custom-designed camera and video encoder architecture are shown in Figure 6.5. In this figure, the following items can be seen. For the 2nd generation video camera (top row), the whole camera acts as a passive heat sink, with the main processor thermally connected to the housing through a heat conducting patch (pink patch in the picture). The 1st generation video camera (middle row) has the processing board with the DM642 DSP on top, with the CCD sensor electronics on the board below. It can be observed that the processing board of the 1st generation camera is identical to the processing board of the 1st generation video encoder (bottom row). This video encoder employs a different board that digitizes the analog video signal, connected to the left of the processing board.

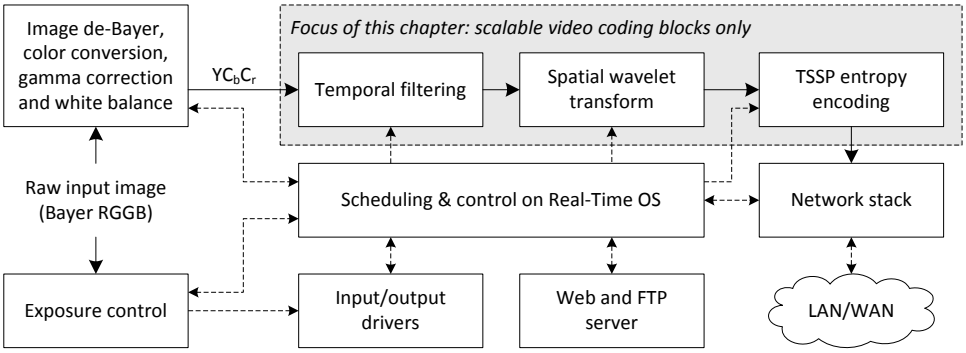


Figure 6.4: Software architecture of the video surveillance camera. Solid arrows indicate video data, dotted arrows refer to control and information data. The grey block embodies the content of this chapter.

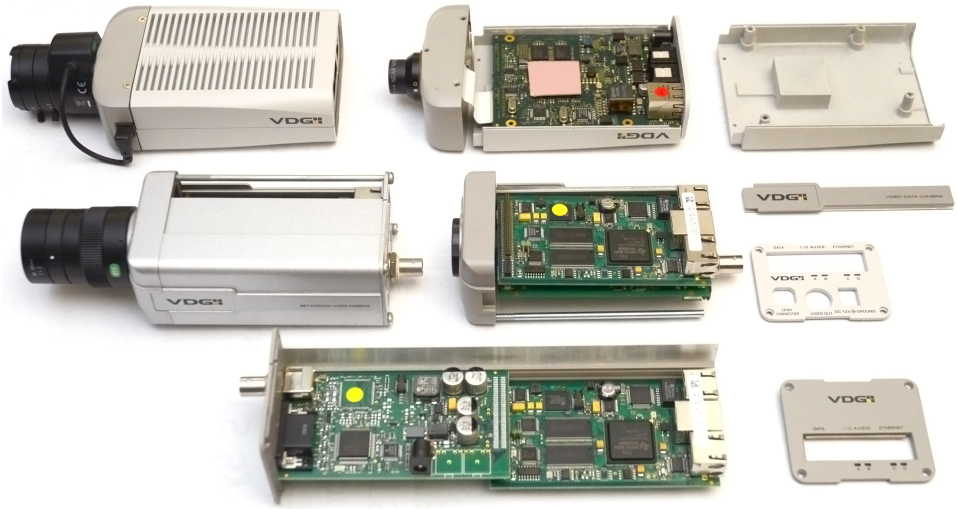


Figure 6.5: Custom-designed hardware architectures and development platforms used for development and profiling. Top row: 2nd generation video camera with advanced multimedia hardware accelerators. Middle row: 1st generation video camera with DM642 processor. Bottom row: 1st generation video encoder with DM642 processor.

Besides the hardware components, a computer with software is utilized to obtain real-time information of the processor status through a JTAG interface (not shown in the figure). By carefully reading the status of hardware counters, profiling of software modules can be performed, so that implementation or algorithmic bottlenecks can be found.

6.3 Discrete Wavelet Transform (DWT) implementation

Introduction

As the architecture of the video surveillance camera and its DM642 Digital Signal Processor (DSP) have been presented, the first important topic for implementation is the mapping of the spatial wavelet transform on the DSP.

The computational complexity of wavelet filters can be reduced by applying the lifting framework and replacing the floating-point wavelet filters by custom-designed integer-to-integer wavelet transforms (integer wavelets in short), as discussed in Section 2.3. Adams and Kossentini [109, 110] have compared the performance of such integer wavelet transforms. With respect to computational complexity, they concluded that the 5/3 or (2,2) transform proposed by Cohen *et al.* [25], Le Gall and Tabatabai [24] and Calderbank *et al.* [22, 23] has the lowest complexity and performs well in lossless compression for images with a larger amount of high-frequency content. As the 5/3 transform is designed for lossless compression, the low- and high-pass filter components are not balanced properly for lossy image compression. By modifying the transform and adding integer scaling factors, such as proposed by Zhang Li-Bao and Wang Ke [31], it is possible to maintain a lossless transformation and improve rate-distortion performance for lossy compression.

We have adopted the 5/3 integer-to-integer lifting framework for our DSP (Digital Signal Processor) implementation, as it can be mapped efficiently to the DSP's VLIW (Very Long Instruction Word) parallel core, while providing good lossless and lossy compression (with the integer scaling factors). Looking at implementation aspects of these wavelet transforms, Chatterjee and Brooks [111] and Meerwald *et al.* [112] have changed the behavior of the transform to improve the use of automatic cache memories in various architectures, but they did not employ architecture-specific optimizations. By utilizing DMA and SIMD, Choi *et al.* [113] claim real-time performance on an embedded system, however, without discussing multi-level transforms. Therefore, this section proposes a novel way of advanced cache management for multi-level spatial wavelet transforms, based on DMA and SIMD-optimized filter kernels.

Lifting framework and the 5/3 integer wavelet

To facilitate the reading of this chapter, this section summarizes the explanation of the lifting framework and its application to the 5/3 integer wavelet from Section 2.3. In Chapter 2, it is explained that when using the lifting framework [28], the wavelet can be implemented with less multipliers and adders than the straightforward FIR implementation. Figure 2.3 shows the principle of the lifting framework. Input samples are split into odd and even samples, after which the even samples are filtered and used to adjust the odd samples in the predict step. Likewise, the odd samples are then filtered and used to adjust the even samples in the update step. Finally, a multiplication is performed to balance the signal energy between low-pass and high-pass output.

Figure 6.6 shows a single combination of an update and a predict step, which is sufficient to implement the 5/3 wavelet filter. For this filter, the predict and update steps are defined by Equations (6.1) and (6.2), respectively, which are specified by

$$x[n] = x[n] - \lfloor (x[n-1] + x[n+1])/2 \rfloor \text{ for } n \bmod 2 = 1, \tag{6.1}$$

$$x[n] = x[n] + \lfloor (x[n-1] + x[n+1])/4 \rfloor \text{ for } n \bmod 2 = 0. \tag{6.2}$$

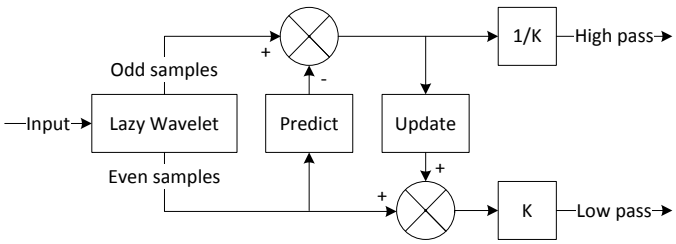


Figure 6.6: Principle of lifting framework for wavelet filtering. The lazy wavelet means the splitting of odd and even samples in the transform.

General implementation aspects

As can be derived from Equations (6.1) and (6.2), the multiplication factors in the predict and update steps are powers of two, which results in filter coefficients for the 5/3 lifting implementation, restricted to 1/8, 1/4, 1/2 and 2. In computing architectures these multiplications can be easily implemented using bit shifting. Rounding of the results is established using the floor operator, which automatically occurs when using the shift function, while also the round operator can be

implemented efficiently, by including an addition with half the division value, followed by the shift operator.

To calculate the output, data from surrounding pixels is necessary. For the 5/3 wavelet filter, the data needed from surrounding pixels involves only a few pixels about the central pixel. Therefore, pre-fetching of data using DMA is possible, but the double use of values across memory transfers should be taken into account. Furthermore, data is also read in a very regular manner and calculations can be rewritten to use basic operations (add, shift, etc) with great locality. By carefully designing the filtering steps, it is possible to effectively use SIMD instructions to perform multiple calculations in a single clock cycle.

The following sections examine the optimal use of SIMD and DMA to implement the 2D filtering process. The discussion is separated into subsections, starting with horizontal filtering and then vertical filtering. Once the discussion about filtering is completed, the cache management for these filtering actions is investigated. Finally, the dyadic multi-level structure is examined. All optimization steps utilize `int16` wavelet coefficients and a standard SD image size of 704×576 pixels.

Horizontal filtering using SIMD

For the horizontal filtering, it is assumed that a line of input pixels is present and a line of output pixels is required. The 5/3 wavelet is implemented using lifting with separate predict and update passes, where a symmetric boundary extension is explicitly implemented. After filtering, the low- and high-pass output samples are still interleaved and need to be split into two separate low- and high-pass bands.

Predict step

The first part of the horizontal wavelet filtering is the predict step. This predict step calculates the high-pass output and stores the results back in the input buffer, following Equation (6.1). To facilitate a very efficient calculation, regular pixel-intensive computations (bulk processing) are split from the symmetric extensions through partitioning the calculation into *prolog*, *kernel* and *epilog* stages. Bulk processing is performed in the *kernel*, of which a graphical representation is given in Figure 6.7.

In this figure, the italic names indicate registers: the *a*, *b* and *out* registers have a width of 64 bits and contain 4 pixels. Similarly, the *tmp* registers have a width of 32 bits and contain 2 pixels. In the diagram, the 64-bit registers are represented by 4 blocks, one for each 16-bit signed value. The 32-bit registers are represented by 2 blocks in a similar fashion. The data labels in these blocks refer to the data content and the calculated values, thereby illustrating the calculation of the predict step in Equation (6.1). The computation of the *prolog*, *kernel* and *epilog* stages is also given

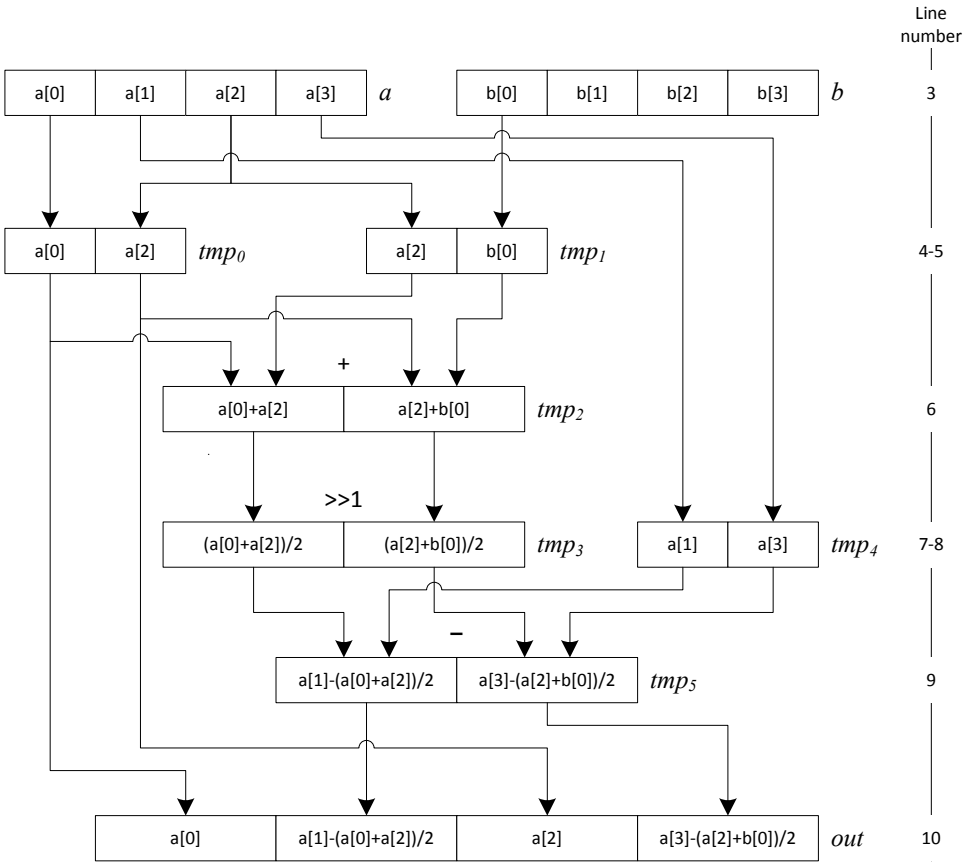


Figure 6.7: Computation flow of the *kernel* of the horizontal predict step, where the blocks represent register contents.

in pseudo-code in Algorithm 6.1, where register naming in the pseudo-code and Figure 6.7 are consistent. For the implementation, the architecture's 64-bit load and store paths are utilized, combined with parallel processing on the 32-bit ALUs.

The *kernel* of the computations is visualized in Figure 6.7. The *kernel* performs the core of the processing, where it automatically reads upcoming data using the special 64-bit transfers until the boundary extension has to be performed. The filter overlap with the previous 64-bit data segment is taken care of by storing the old input values in a register, so that duplicate external memory access is not required. Following the figure from top to bottom, a short explanation of each of the steps is given in Table 6.2. The indicated line numbers correspond to the line number of the pseudo code of Algorithm 6.1 and are also given at the right side of Figure 6.7.

Table 6.2: Explanation of horizontal predict step *kernel* algorithm

Line	Description
3	Read the next 4 pixels into register b
4–5	Extract even samples $a[0]$, $a[2]$, $b[0]$, store in tmp_0, tmp_1
6	Add even samples in tmp_0 and tmp_1, store in tmp_2
7	Shift right 16-bit parts of tmp_2, resulting in $\lfloor x \div 2 \rfloor$
8	Extract odd samples $a[1]$ and $a[3]$, store in tmp_4
9	Subtract 16-bit parts of tmp_3 and tmp_4, completing the predict step, store in tmp_5
10	Combine even samples $a[0]$ and $a[2]$ and the results from the lifting step, store in *out*
11	Write *out* back to location where a was read from.
12	Assign $a = b$. The data read in line 3 will be processed in the next iteration

The *prolog* and *epilog* are used to perform preparation and clean-up for the fast *kernel* implementation, while simultaneously being used to perform the symmetric extension. For the *prolog*, the first 4 pixels are read, which takes care of initializing the data buffers (Line 1 in Algorithm 6.1). The read uses a special 64-bit data transfer to communicate all four values at once, so that the full width of the memory bus is exploited without waste. For the predict step, symmetric image extension can be omitted at this point.

In the *epilog* (Lines 14–21), the symmetric image extension is explicitly implemented to avoid boundary effects from the transform. Instead of reading the next 4 pixels and extracting $b[0]$, the image is folded around $a[3]$. This results in the use of a mirrored inside pixel $a[2]$, instead of outside data $b[0]$. The remainder of the calculations is identical to those in the *kernel*.

As can be observed from this description, the algorithm is specifically designed to avoid waste of memory bandwidth, so that every wavelet coefficient is only read once, even though some wavelet coefficients are used multiple times in the calculations. An example of the optimized code is given in Appendix F.

Update step

The second part of the horizontal wavelet filtering is the update step. This update step calculates the low-pass output and places the results back in the input buffer, following Equation (6.2). For ease of understanding, both wavelet filtering equations are reproduced here to illustrate their similarity, with the predict and update

Algorithm 6.1 Horizontal predict step algoritm

```
    ↷ Prolog
 1: a = read 64-bit from in[0]

    ↻ Kernel
 2: for x = 0 to width − 4, step 4 do
 3:     b = read 64-bit from in[x + 4]
 4:     tmp0 = retrieve even samples: a[0] and a[2]
 5:     tmp1 = retrieve even samples: a[2] and b[0]
 6:     tmp2 = add 16-bit parts of tmp0 and tmp1
 7:     tmp3 = shift right both 16 bit parts of tmp2
 8:     tmp4 = retrieve odd samples: a[1] and a[3]
 9:     tmp5 = subtract 16-bit parts of tmp3 from tmp4
10:     out = put tmp0 in even and tmp5 in odd positions
11:     write 64-bit out to in[x]
12:     a = b
13: end for

    ↶ Epilog
14: tmp0 = retrieve even samples: a[0] and a[2]
15: tmp1 = retrieve even samples: a[2] and a[2]
16: tmp2 = add 16-bit parts of tmp0 and tmp1
17: tmp3 = shift right both 16 bit parts of tmp2
18: tmp4 = retrieve odd samples: a[1] and a[3]
19: tmp5 = subtract tmp3 from tmp4
20: out = put tmp0 in even and tmp5 in odd positions
21: write 64-bit out to in[width − 4]
```

steps shown in Equations (6.3) and (6.4), respectively, which are specified by

$$x[n] = x[n] - \lfloor (x[n-1] + x[n+1])/2 \rfloor \text{ for } n \bmod 2 = 1, \tag{6.3}$$

$$x[n] = x[n] + \lfloor (x[n-1] + x[n+1])/4 \rfloor \text{ for } n \bmod 2 = 0. \tag{6.4}$$

From these equations, it can be observed that the update step is very similar to the predict step. Therefore, a detailed breakdown, such as given for Figure 6.7 and Algorithm 6.1, is omitted here.

The implementation of the update step is listed in pseudo-code in Algorithm 6.2, where the following differences with the update step are noticed. The symmetric extension is now implemented in the *prolog* (Lines 1–9), instead of the *epilog*. The *kernel* (Lines 10–21) performs the update step with a minor adjustment in the calculation (subtract replaced by add and different number of bit-shifts). The *epilog* stage is not required.

Algorithm 6.2 Horizontal update step algorithm

```
    ↷ Prolog
 1: a = read 64-bit from in[0]
 2: tmp0 = retrieve odd samples: a[1] and a[1]
 3: tmp1 = retrieve odd samples: a[1] and a[3]
 4: tmp2 = add 16-bit parts of tmp0 and tmp1
 5: tmp3 = shift right 2 both 16 bit parts of tmp2
 6: tmp4 = retrieve even samples: a[0] and a[2]
 7: tmp5 = add tmp3 to tmp4
 8: out = put tmp0 in even and tmp5 in odd positions
 9: write 64-bit out to in[0]

    ↻ Kernel
10: for x = 4 to width, step 4 do
11: |  b = read 64-bit from in[x]
12: |  tmp0 = retrieve odd samples: a[3] and b[1]
13: |  tmp1 = retrieve odd samples: b[1] and b[3]
14: |  tmp2 = add 16-bit parts of tmp0 and tmp1
15: |  tmp3 = shift right 2 both 16 bit parts of tmp2
16: |  tmp4 = retrieve even samples: b[0] and b[2]
17: |  tmp5 = add tmp3 to tmp4
18: |  out = put tmp0 in even and tmp5 in odd positions
19: |  write 64-bit out to in[x]
20: |  a = b
21: end for
```

Splitting step

The third and last part of the horizontal wavelet filtering is the splitting step. The predict and update stages discussed in the previous parts, result in an output data that interleaves the low- and high-pass data. The splitting step is required to split this interleaved data into two separate low- and high-pass bands. This step is fully performed employing 8-tuples of samples, leading to the exclusive use of SIMD instructions, as illustrated by the pseudo-code in Algorithm 6.3.

Vertical filtering using SIMD

To complete the 2D wavelet transform, vertical filtering is applied after the horizontal filtering. Because the coefficients are stored per line, vertical filtering is efficiently implemented when processing complete lines and vertical processing can already start when enough horizontally-filtered lines have been created. This quick start also facilitates efficient use of cache memories, because the horizontally filtered data can move straight to the vertical filtering processing, without intermediate storage in external memory.

Algorithm 6.3 Horizontal splitting step algorithm

```
   ↻ Kernel
1: for x = 0 to width, step 8 do
2:     in_0 = read 64-bit from in[x]
3:     in_1 = read 64-bit from in[x + 4]
4:     out_0 = retrieve even samples from in_0 and in_1
5:     out_1 = retrieve odd samples from in_0 and in_1
6:     write 64-bit out_0 to out[x/2]
7:     write 64-bit out_1 to out[x/2+width/2]
8: end for
```

Vertical filtering is performed on three already horizontally filtered lines. Algorithm 6.4 shows the use of SIMD instructions in the vertical predict step. All registers have a width of 64 bits and pointers to lines A, B and C are given as input. Lines A and C are read prior to line B, as they are used first in the calculations, thereby nullifying the 4-cycle wait state of the memory transfer. Similar to horizontal filtering, the update step is nearly identical to the predict step, except for a shift right of two bit positions in Line 6 and an addition instead of the subtraction in Line 7.

The symmetric extension, management of odd and even lines and low-/high-pass splitting are processed outside the vertical filtering loop through smart background DMA transfers. The background DMA transfers and their role in vertical symmetric extension is discussed in Section 6.3.

Algorithm 6.4 Vertical Predict Step

```
   ↻ Kernel
1: for x = 0 to width, step 4 do
2:     a = read 64-bit from &LineA[x]
3:     c = read 64-bit from &LineC[x]
4:     b = read 64-bit from &LineB[x]
5:     tmp_0 = add 16-bit parts of a and c
6:     tmp_1 = shift right 16 bit parts of tmp_0
7:     out = subtract 16-bit parts of tmp_0 from b
8:     write 64-bit out to &LineB[x]
9: end for
```

Cache management for 2D wavelet filtering

Up to this point, the fast implementation of the computational parts of the horizontal and vertical wavelet filtering have been discussed. These calculations utilize

coefficient data from the external memory as input and write the calculated output back to external memory. The DM642 contains an automatic caching strategy to mirror frequently-used memory segments in the Level-2 (L2) cache. However, this automatic caching can be sub-optimal if sudden changes in data organization occur.

Since the pixel-access patterns for the wavelet transform are very structured, we have chosen to explicitly manage a part of the available cache memory ourselves using DMA techniques, with the aim to fully exploit parallel data computations and memory transfers. In total, 7 image lines for various buffers are reserved in the L2 cache, using $7 \times 704 \times 2 = 9,856$ Bytes. These buffers are used for horizontal filtering, vertical filtering and input/output buffering.

In order to avoid continuously moving coefficient data and intermediate data around, a system using 9 data pointers is employed. Two pointers are defined for the current and target images in the external memory, pC and pT, respectively. The pointers $pLineX$ and $pLineY$ are used for the ping-pong buffering system of the horizontal filter and shown in Figure 6.8. The pointers are used for loading input lines from external memory and as a scratch buffer for the horizontal filtering. Five pointers to memory locations in the L2 cache are used to represent the lines that were horizontally filtered, $pLineA$–$pLineE$. The pointers are rotated (indicated by the dotted lines) and used for supplying new lines from horizontal filtering, performing vertical filtering and moving the results to external memory.

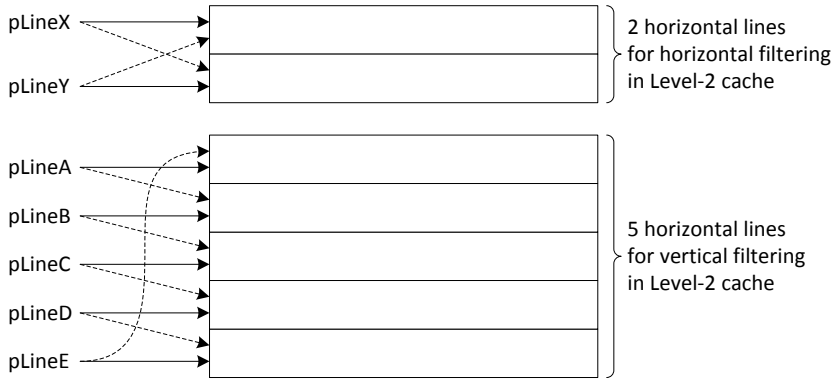


Figure 6.8: Video line buffers for horizontal and vertical filtering in the L2 cache. The dotted arrows indicate the rotation order.

For 2D wavelet filtering, two DMA channels are used simultaneously. The high-priority channels are one-dimensional and exploited for moving image lines in and out of the cache: DMA_{In} and DMA_{Out}.

The cache management algorithm is depicted in pseudo-code in Algorithm 6.5 and split in *prolog*, *kernel* and *epilog* stages, similar to the wavelet filtering algorithms. In the *prolog*, pointers are initialized (Lines 1–2) and the first image line is copied to $pLineX$ using DMA_{In}. In Lines 4–9, consecutive image lines are copied using DMA_{In} and simultaneously filtered horizontally, using the algorithms described in Section 6.3. After this loop has finished, the L2 cache memory contains the data in the order as shown in the second column of Table 6.3, which is the minimum required input data for the vertical filtering process.

Table 6.3: Contents of Level-2 cache at various moments in time. The following shorthand notation is used: H.Filt = Horizontally filtered and split, V.Filt = Vertically filtered line, LPx = Low-Pass result number x, HPx = High-Pass result number x, Temp.H.Filt x = Temporary data from horizontal filtering of line x (interleaved data).

Cache line	Time = *prolog* prior to V.Filt	Time = just prior to V.Filt in *kernel* loop with y=2	y=3	y=4
0	Empty	Line 4 H.Filt	Line 4 H.Filt	Line 4 H.Filt
1	Line 0 H.Filt	V.Filt LP0	Line 5 H.Filt	Line 5 H.Filt
2	Line 1 H.Filt	V.Filt HP1	V.Filt HP1	Line 6 H.Filt
3	Line 2 H.Filt	Line 2 H.Filt	Line 2 H.Filt	V.Filt LP3
4	Line 3 H.Filt	Line 3 H.Filt	V.Filt HP3	V.Filt HP3
5	Input line 4	Temp.H.Filt 4	Input line 6	Temp.H.Filt 6
6	Temp.H.Filt 3	Input line 5	Temp.H.Filt 5	Input line 7

In Line 10, the vertical predict filter is executed, which stores its output to the line indicated by $pLineC$ (vertical filtering was discussed in Section 6.3). The resulting high-pass output is then transferred to the output image pT using DMA_{Out} to image line $h/2$ in Line 11. Line 12 executes the vertical update filter with symmetrical boundary extension, by providing $pLineC$ as the first input instead of $pLineA$. The resulting low-pass output is then transferred to the output image pT using DMA_{Out} to image Line 0 in Line 14. It can be clearly seen that the vertical splitting is performed implicitly through the DMA_{Out}.

After the initialization and symmetric extension in the *prolog*, the rest of the image is processed in the *kernel*. The *kernel* rotates the pointers, waits for the DMA_{I} to finish and executes the horizontal filtering in Lines 16–18. If new data is still present (Line 19), DMA_{In} is started to load this line into the L2 cache in Line 20. Then, alternating each image line, either a predict step (Lines 23–25), or an update step (Lines 27–29) is performed. From the targets in Lines 25 and 29, it can be clearly seen that the vertical splitting is performed implicitly through the DMA_{Out}.

Algorithm 6.5 Cache management for 2D wavelet filtering

↷ *Prolog*
1: Initialize *pLineA*–*pLineE* to L2 cache line 0–4
2: Initialize *pLineX*–*pLineY* to L2 cache line 5–6
3: Start DMA_{In} to copy *pC* line 0 to *pLineX*
4: **for** $y = 1$ to 4 **do**
5: | Wait for DMA_{In} to finish
6: | Swap *pLineX* and *pLineY*
7: | Start DMA_{In} to copy *pC* line y to *pLineX*
8: | Horizontally filter *pLineY* and place in cache line y
9: **end for**
10: Vertically filter predict (*pLineB*,*pLineC*,*pLineD*)
11: Start DMA_{Out} to copy *pLineC* to *pT* line $h/2$
12: Vertically filter update (*pLineC*,*pLineB*,*pLineC*)
13: Wait for DMA_{Out} to finish
14: Start DMA_{Out} to copy *pLineB* to *pT* line 0

↻ *Kernel*
15: **for** $y = 2$ to $h - 2$ **do**
16: | Rotate pointers *pLineA*–*pLineE*
17: | Wait for DMA_{In} to finish
18: | Horizontally filter *pLineX* and place result in *pLineE*
19: | **if** $y < h - 3$ **then**
20: | | Start DMA_{In} to copy *pC* line $y + 3$ to *pLineX*
21: | **end if**
22: | **if** $y \bmod 2 == 0$ **then**
23: | | Vertically filter predict (*pLineC*,*pLineD*,*pLineE*)
24: | | Wait for DMA_{Out} to finish
25: | | Start DMA_{Out}: *pLineD* to *pT* line $y/2 + h/2$
26: | **else**
27: | | Vertically filter update (*pLineA*,*pLineB*,*pLineC*)
28: | | Wait for DMA_{Out} to finish
29: | | Start DMA_{Out}: *pLineB* to *pT* line $y/2$
30: | **end if**
31: **end for**

↶ *Epilog*
32: Vertically filter predict (*pLineD*,*pLineE*,*pLineD*)
33: Wait for DMA_{Out} to finish
34: Start DMA_{Out} to copy *pLineE* to *pT* line $h - 1$
35: Vertically filter update (*pLineC*,*pLineD*,*pLineE*)
36: Wait for DMA_{Out} to finish
37: Start DMA_{Out} to copy *pLineD* to *pT* line $h/2 - 1$

The contents of the cache memory during the *kernel* processing are given in Table 6.3 in the last three columns. The indicated data is present after the input line is horizontally filtered, just prior to filtering a vertical line. It can be observed that during vertical high-pass processing ($y \pmod 2 \equiv 0$), just enough cache memory is utilized to store the horizontally-filtered lines required for vertical high-pass processing. For vertical low-pass processing ($y \pmod 2 \equiv 1$), just sufficient vertical high-pass lines and the remaining horizontally-filtered line are in the cache memory, which are required to calculate the vertical low-pass result. This process alternates between vertical low- and high-pass processing, until the last line in the image is read and horizontally filtered, after which the *epilog* processing stage starts.

At this point, horizontal filtering has fully completed. In the *epilog*, the last lines are vertically filtered and symmetrical boundary extension is performed. The boundary extension is effectively implemented by providing *pLineD* as the third input in Line 32 and *pLineE* as the third input in Line 35. The final two vertical filtering splits are performed in Lines 34 and 37 using DMA_{Out}.

Multi-level 2D DWT reconstruction using DMA

The 2D wavelet filtering described in Sections 6.3, 6.3 and 6.3 only performs a single iteration of the dyadic multi-level 2D wavelet decomposition. In this section, the optimized 2D wavelet filtering of the previous sections is expanded to a full dyadic multi-level decomposition.

In addition to the two high-priority DMA channels utilized for horizontal and vertical wavelet filtering, three additional low-priority DMA channels are used for the dyadic multi-level 2D wavelet decomposition. All five DMA channels are used simultaneously and automatically scheduled according to their priority. The three low-priority DMA channels for reconstructing the 2D multi-level image are 2D in nature and denoted by: DMA_{A}, DMA_{B} and DMA_{C}. These 2D channels are able to move rectangular image blocks from one location to the other, while supporting different line pitch[1] between the source and the target.

At the end of each level, the image pointers *pC* and *pT* are swapped, the level width and height is divided by two and the 2D wavelet transform process is restarted. For a 4-level transform, this process is illustrated in Figure 6.9, with the wavelet filter operating on the data associated with the grey parts of the buffers. In the figure it can be seen that because of the alternating 2D wavelet filtering, neither of the two buffers will contain the complete dyadic decomposition. Therefore,

[1] A line pitch is the line width for communication, but rounded up to the N-tuple of the communication word length, e.g. 64 bits.

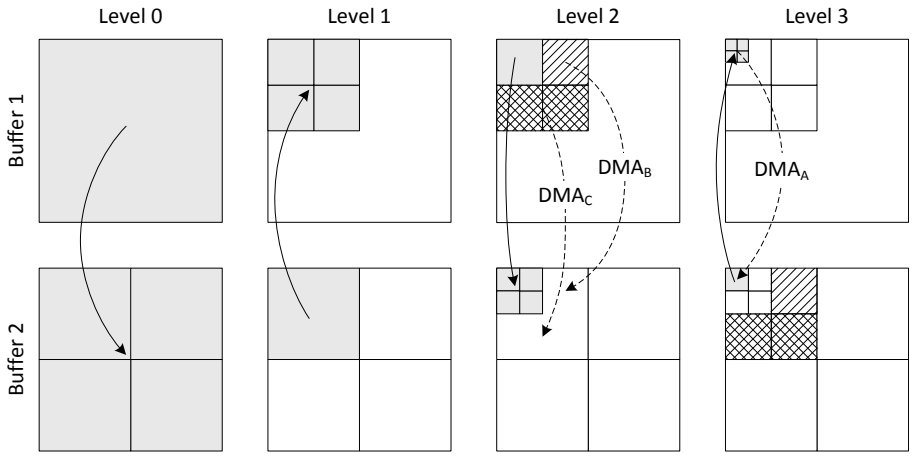


Figure 6.9: Multi-level reconstruction using DMA. Wavelet filtering is indicated by the solid arrows and DMA transfers by the dotted arrows with the channel name in subscript.

DMA channels DMA_B and DMA_C are employed to move the top-right quadrant and bottom-half of the wavelet image from Buffer 1 to Buffer 2, once every two hierarchical levels. This is indicated in the figure by the dotted arrows. In case that an even number of levels is desired, then also the last output has to be copied from Buffer 1 to 2 using DMA_A after the last wavelet transform. Channels DMA_{A-C} execute at a lower priority than the line in- and output channel DMA_{In} and DMA_{Out}, thereby not disturbing these transfers.

This process of background DMA transfers is so effective, that the ALUs are always supplied with input data. At the fourth level, there is a small wait interval for the DMA_B and DMA_C channels to complete their tasks, due to the small size of the image to be transformed at that moment. For an even number of levels, an additional DMA_A transfer is started to copy the last output to Buffer 2. However, the wait cycles for these transfers are insignificant, due to the small sizes of the copy operation. Furthermore, the wait cycles can be nullified completely by starting other calculations on the DSP and allowing the DMA to complete the copying as background processing.

6.4 Two-Stage SPECK (TSSP) implementation

Now that the 5/3 integer wavelet transform is fully optimized for the video surveillance architecture, similar techniques are employed to implement the more complex and more dynamic TSSP wavelet coefficient encoder in this section.

Optimization possibilities and caveats

The TSSP algorithm from Chapter 3 is specifically designed for implementation on embedded systems. Part of this design is the decision to split the algorithm in two stages as explained in Chapter 3. It is clear these two stages offer different optimization and implementation possibilities. Since the processing in Stage 1 is data-independent, processing can be fully streamlined, so that advanced background retrieval of data can be exploited. In Stage 2, optimization possibilities arise from the fact that all bit planes are processed in parallel and that coefficients for coding are in close spatial proximity. A common bottleneck in video coding systems arises from bit-by-bit bitstream creation during entropy coding, which needs to be addressed in Stage 2 as well. Finally, the temporary Significance Level (SL) buffer is critical for both stages and should therefore be kept in fast memory.

As discussed in Section 3.3, the quadtree has a fixed depth ($N_{QTdepth}$) and size (N_{QTelem}) for a given image size. The resulting required buffer size for the whole SL buffer is displayed in Table 6.4, for various common image resolutions. Storage requirements using both `uint4` and `uint8` elements, are given, because for wavelet coefficients of `int16` precision, the significance level of the quadtree nodes can be stored in an `uint4` element.

Table 6.4: Size of TSSP quadtree and Significance Level (SL) buffer.

Size \ Resolution	1920×1088 [pixels]	1280×720 [pixels]	704×576 [pixels]	704×480 [pixels]
Quadtree: $N_{QTdepth}$	9	8	8	7
Quadtree: N_{QTelem}	349,525	87,381	87,381	21,845
SL buffer (`uint4` elements)	171 kB	42.7 kB	42.7 kB	10.7 kB
SL buffer (`uint8` elements)	341 kB	85.3 kB	85.3 kB	21.3 kB

The optimizations in this section are carried out for a standard 4CIF image resolution of 704×576 pixels and wavelet coefficients of `int16` precision. Formatting has been chosen such that sub-Byte access is prevented.For a single frame, these parameters result in the data transfers towards and in between the two stages, as depicted in Figure 6.10. The SL buffer is created in Stage 1 and read in Stage 2 for the significant blocks only. The small size of this buffer (85.3 kB, Table 6.4) indicates that it can be included in fast Level-2 cache memory. Furthermore, by processing all bit planes in parallel, the input image is read once within Stage 1 and only partly addressed in Stage 2, thereby significantly reducing memory bandwidth.

The applied optimizations for Stage 1 and Stage 2 are discussed in Sections 6.4

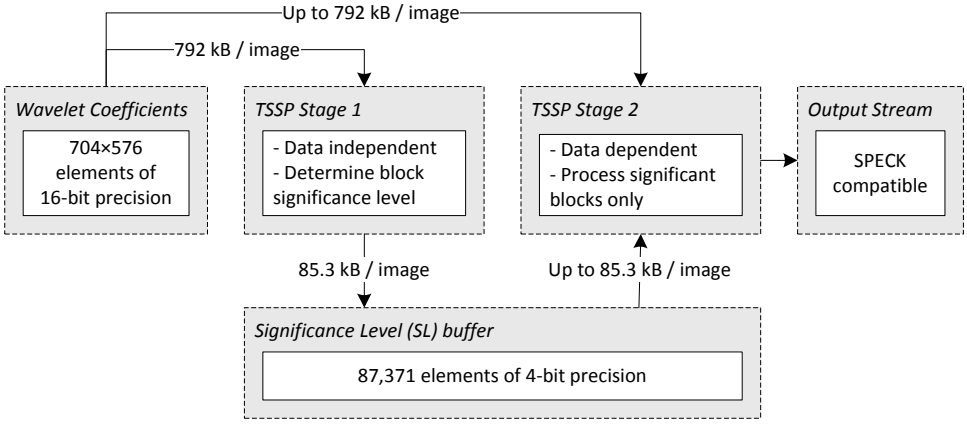


Figure 6.10: Schematic diagram of Two-Stage SPECK (TSSP) with data transfers and buffer sizes for a image of 704×576 pixels.

and 6.4, respectively. For clarity, both discussions address two topics of interest: memory optimizations and data and code optimizations. Section 6.4 presents the aspect of memory management and bandwidth limitations for both stages.

Optimization of Stage 1

Stage 1 of TSSP is used to calculate the Significance Level (SL) of all nodes in the quadtree, which are then stored in the SL buffer. Since the SL of a node can be directly derived from the SL of its four leafs, the SL buffer is generated backwards, scanning the image from bottom-right to top-left in a reversed Morton-order pattern. The algorithmic description of Stage 1 can be found in Section 3.3.

Memory optimizations

Two-dimensional DMA transfers are employed to move square blocks of 88×72 coefficients from the external memory to the Level-2 cache, 1/64th of the size of a 704×576 image. The width of these blocks is chosen to be a multiple of 8 Bytes, so that the 64-bit physical connections between external and cache memory can be fully exploited. A ping-pong buffer scheme is used, so that memory transfers of the next block already occur while performing calculations on the current block. The output of Stage 1, the SL buffer, is also placed in the Level-2 cache, for fast write access in Stage 1 and fast read access in Stage 2.

Data and code optimizations

Blocks of size 11×9 coefficients are processed as one entity, as the quadtree partitioning becomes irregular after this point. Figure 6.11 visualizes the 21 quadtree nodes and the segmentation of the 11×9 coefficient block.

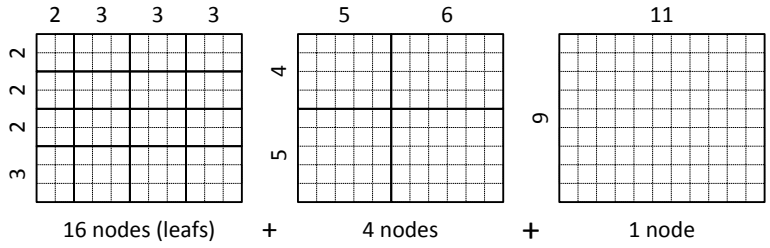


Figure 6.11: The 21 quadtree nodes of the 11x9 coefficient block.

Within this 11×9 coefficient block, the SL parameter is calculated in parallel for multiple coefficients and levels of the quadtree, simultaneously. First, two lines of coefficients are read in chunks of 12 × 2 coefficients using 64-bit transfers, representing four of the leafs. This 2-line memory transfer is depicted in Figure 6.12(b), with the coefficient naming convention as defined in Figure 6.12(a).

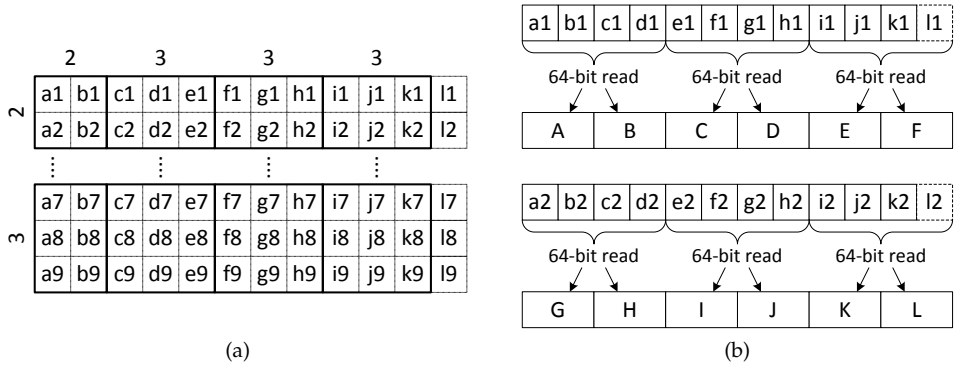


Figure 6.12: Reading operations for the two line transfer, (a) naming convention of the 11x9 data blocks and (b) optimized 64-bit reading of coefficients in kernel using 12×2 region.

Calculations on the 11×9 coefficients block are performed using SIMD instructions and visualized partly in Figure 6.13. First, the absolute value of each coefficient is determined, after which the two lines are combined using the `OR` operator, which does not affect the position of the first significant bit. Then the left-most bit

(lmb) is calculated and the `uint8` results are re-ordered and packed in `uint32` registers G,H and I. The 12th coefficient is discarded and replaced by the number 32, which is the outcome of the lmb operator on a coefficient of value 0. The combined lmb of all 22 coefficients are obtained separately for each of the four leafs in J, by performing a Byte-wise minimum operation on G,H and I. The SL of the four leafs is calculated by $SL = 32 - lmb$, followed by a unity subtraction for values > 0 and stored temporary. The conditional subtraction is implemented without any '`if`' statements, using Byte-wise 'subtract' and 'minimum' operations.

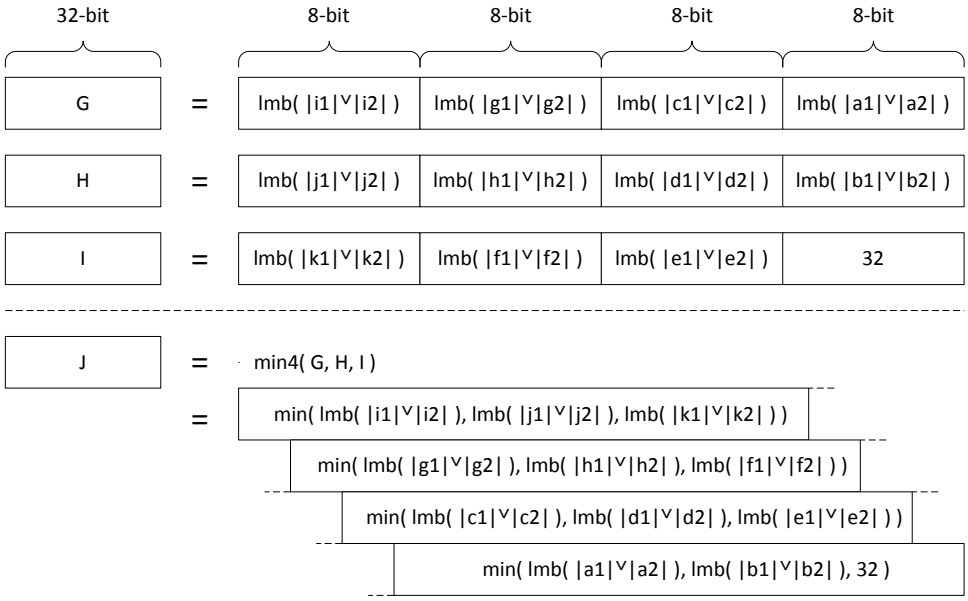


Figure 6.13: 2-Line calculation kernel for 4 quadtree nodes in parallel, using SIMD instructions such as lmb (left-most bit), abs, or and min.

The next group of 2 lines is processed identically, giving another 4 SL parameters. Using the now 8 calculated leaf values, the 2 SL parameters of the 2 quadtree groups encompassing these leafs can be calculated. All 10 calculated values are stored in the appropriate order in the SL buffer, using 2 Byte-writes and 2 word-writes, to store all 10 results for the 8 leafs in parallel. The third set of 2 lines is processed and the intermediate SL parameters are stored. The last 4 leafs contain 3 lines instead of 2, but are processed in the same optimized way. When these last 8 leaf SL parameters are obtained, the 2 parent SL parameters are calculated and all parents are combined to obtain the final SL parameter.

Optimization of Stage 2

Stage 2 of TSSP is used to perform the actual encoding process, using the SL buffer created in Stage 1 for decision making. Based on the required quality and the values in the SL buffer, a choice is made to either encode a certain block, or skip the whole underlying tree completely. For more details, the algorithmic description of Stage 2 can be found in Section 3.3.

Memory optimizations

In Stage 2, coefficients are encoded for all bit planes in parallel. The bitstream for each bit plane is stored in external memory, but to avoid the inefficient writing of single bits to external memory, 32-bit write buffers are employed in the Level-2 cache for all 30 sorting and refinement buffers. These 32-bit buffers are explicitly implemented as temporary buffers to facilitate the parallel processing of all bit planes and the sorting and refinement passes in parallel. Very efficient single bit access is accomplished by initializing the buffers to 0, and by using special instructions so that only ones are actually written. All bitstream parts are word-aligned to facilitate fast copying.

Data and code optimizations

Most data and code optimizations concentrate on the coding of individual coefficients. Memory reads are combined for the 2×2, 2×3, 3×2 and 3×3 leafs. Two coefficients are processed in parallel to fully exploit the 32-bit ALUs. Special bit-based writing is possible for 2 and 3 bits in one operation.

Memory management and TSSP bandwidth modeling

A. Memory management

Because of its high complexity in memory lists and transfers of TSSP, we present here an overview of the most important buffers and communication bandwidth. Figure 6.14 shows the location of the buffers in external memory and Level-2 cache. Arrows indicate read and write directions, while buffer sizes are listed in Table 6.5. From the 256-kB Level-2 cache available in the DM642 processor, 128 kB is self-managed, of which 110.3 kB is used by TSSP.

From this figure, it can be observed that a significant amount of data transfers occur during encoding. For Stage 1, the entire input image in external memory is accessed, while the SL buffer is written to Level-2 cache. During Stage 2: (1) the SL buffer is read, (2) all quadtree leafs with at least 1 significant coefficient are read and (3) the output bitstream is written.

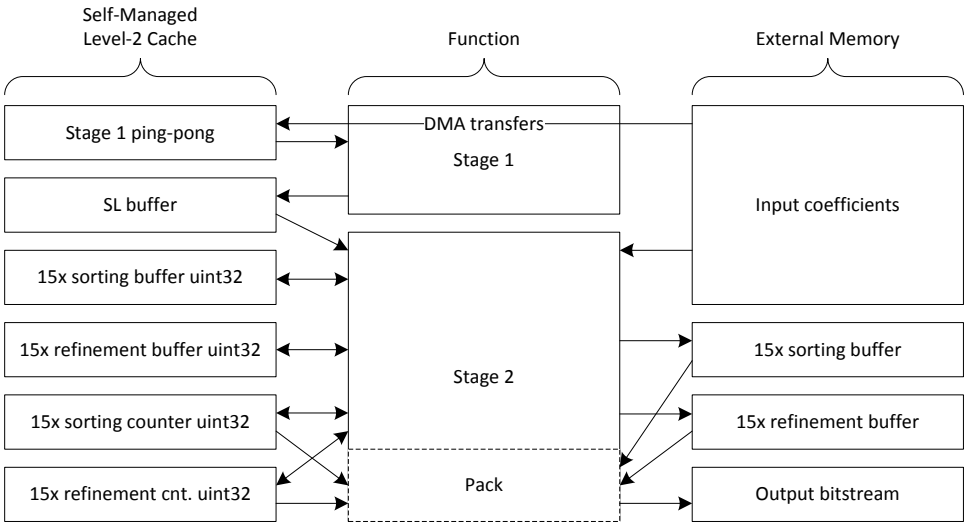


Figure 6.14: Buffer locations and access directions for the self-managed Level-2 cache and external memory in TSSP.

B. TSSP bandwidth modeling

The memory management of the TSSP implementation is so manyfold and complex that a bandwidth estimation model has been developed for management purposes. At the end of this chapter, the developed model explained here will be used for the comparison with actual measurements of the implementation. We define the ideal case for memory transfers when the bandwidth can be fully exploited. For the following development of the model, a transformed 4CIF image is used as a reference case with 405,504 `int16` coefficients, where 10% of the data consists of significant leafs and a typical compression ratio of 10 is obtained. The *SL* buffer contains 87,381 `uint8` elements.

In the used DSP architecture (see Section 6.2, Table 6.1), the external memory has an external memory bandwidth of 2.128 GB/s and the Level-2 cache operates at 9.6 GB/s for reading and writing simultaneously. The estimated times for the main memory transfers of Figure 6.14 are summarized in Table 6.6. The estimations in this table are derived as follows. The image is moved from external memory to Level-2 cache using DMA. These DMA transfers can be parallelized with calculations, which is indicated in the table by the grey shading. These memory transfers involve the division of the amount of coefficients by the external memory

Table 6.5: Buffer sizes for a 4CIF image (704×576 pixels).

External memory	Elements	Precision	Amount	Size [kB]
Image coefficients	405,504	*int*16	1	792
Sorting buffer	76,032	*uint*8	15	1,114
Refinement buffer	38,016	*uint*8	15	557
Output bitstream	405,504	*uint*8	1	396
Total				2,859 kB

Level-2 cache	Elements	Precision	Amount	Size [B]
Stage 1 ping-pong	6,336	*int*16	2	25,344
Significance level	87,381	*uint*8	1	87,381
Sorting write	1	*uint*32	15	60
Sorting count	1	*uint*32	15	60
Refinement write	1	*uint*32	15	60
Refinement count	1	*uint*32	15	60
Total				112,965 Bytes

bandwidth, both mentioned previously. This leads to

$$(405,504 \cdot 2)/2.218 \times 10^9 = 366\,\mu s \quad \text{for Stage 1,} \tag{6.5}$$

$$(0.1 \cdot 405,504 \cdot 2)/2.218 \times 10^9 = 36.6\,\mu s \quad \text{for Stage 2.} \tag{6.6}$$

Table 6.6: Modeled optimal memory transfer times for TSSP. The grey column indicates these transfers are performed using DMA and can be parallelized with calculations.

	Input DMA Ext→L2	Input L2→CPU	*SL* buffer CPU↔L2	Bitstream CPU→Ext	Total
Stage 1	366 μs	169 μs	18.2 μs	N/A	553.2 μs
Stage 2	36.6 μs	16.9 μs	18.2 μs	36.6 μs	108.3 μs
Total	402.6 μs	185.9 μs	36.4 μs	36.6 μs	661.5 μs

The other transfers are performed by the CPU, which cannot be parallelized with calculations. The Level-2 cache transfers of the image is estimated to be

$$(405,504 \cdot 2)/(\tfrac{1}{2} \cdot 9.6 \times 10^9) = 169\,\mu s \quad \text{for Stage 1,} \tag{6.7}$$

$$(0.1 \cdot 405,504 \cdot 2)/(\tfrac{1}{2} \cdot 9.6 \times 10^9) = 16.9\,\mu s \quad \text{for Stage 2,} \tag{6.8}$$

where the factor of a half in the computation results from the fact that only one of the two channels is used for this two-way associative cache. The Level-2 cache transfers of the SL buffer are estimated a different memory transfer cost, giving

$$87,381/(\tfrac{1}{2} \cdot 9.6 \times 10^9) = 18.2\,\mu s, \tag{6.9}$$

for both Stage 1 and Stage 2. Here, writing the output bitstream to the external memory in Stage 2 is estimated as

$$(0.1 * 405,504 * 2)/(2.218 \times 10^9) = 36.6\,\mu s. \tag{6.10}$$

In total, 661.5 μs is estimated for transferring the data, of which 402.6 μs can be parallelized with computations through DMA, while 258.9 μs is performed using CPU cycles. These calculated outcomes of the memory processing model are used in Section 6.6 to evaluate the realism of the applied memory transfer model.

6.5 Temporal filtering implementation

Known limitations of the temporal implementation

For the validation of the temporal filtering, the temporal filtering structure as proposed in Chapter 4 is implemented, albeit without the use of motion estimation and compensation. The development of the HPPS algorithm was performed on an experimental camera platform that did not contain the full software tools for exploiting specific hardware accelerator functions. A new setup with an improved processor architecture was not yet available at the time of completing this research.

Instead, a flexible implementation of the temporal filtering structure has been realized in the utilized test platform, which offers full temporal scalability. Due to the absence of motion estimation and compensation in the temporal tree, the coding efficiency is decreased when compared to the full coding system implementation, but in the case of static video surveillance, this penalty is very small and directly related to the amount of motion in the scene.

Description of the temporal implementation

The temporal filtering is constructed upon a dedicated dynamic memory manager that stores: (1) input frames, (2) output frames ready for entropy coding and (3) intermediate temporal bands. The memory manager is depicted in Figure 6.15, utilizing a similar pointer structure to avoid video frame memory movements. Furthermore, to facilitate efficient DMA use, all communication is aligned to 8-Byte boundaries.

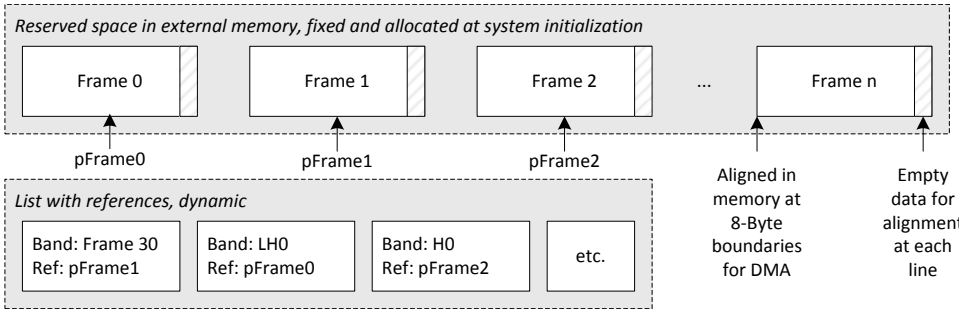


Figure 6.15: Memory manager for DSP implementation of temporal filtering.

The memory manager is controlled by a set of instructions similar to a small computer program. This set of instructions indicates when certain bands need to be processed and when they can be discarded. The set of instructions can be automatically generated from the dynamic analysis of the temporal configuration and the dependencies between bands. The dynamic behavior of the proposed BiDirectional Low Delay (BDLD) configuration, is repeated from Chapter 4 and shown in Figure 6.16. Full temporal filtering is performed (without ME) and a complete video stream is created.

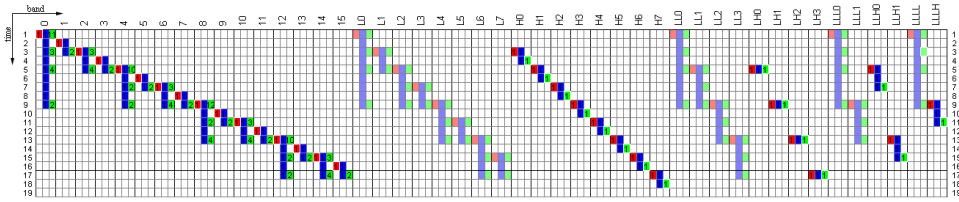


Figure 6.16: Dynamic behavior of the BDLD configuration. A larger version of this figure can be found in Figure 4.9, combined with an explanation in Section 4.3.

All temporal filtering is performed using SIMD instructions, similar to the implementations of the wavelet and TSSP coding algorithm, hence a detailed discussion is omitted. Filtering is performed sequentially for the whole image and the SIMD principle is applied to process multiple pixels in parallel. Due to the straightforward memory access pattern, automatic caching is employed.

6.6 Experimental results and discussion

This section discusses the performance of the optimized implementations, through measurements taken from the real-time embedded system.

Discrete Wavelet Transform (DWT)

The 4-level 5/3 wavelet transform is implemented on a custom-designed video surveillance camera, using the techniques described in this chapter (See Section 6.3). Table 6.7 shows the obtained cycle count for executing each particular level, as well as the total amount of cycles for performing the complete multi-level 2D 5/3 wavelet transform. The table illustrates that a single 4-level transform at 4CIF broadcast resolution requires 3.65 Mcycles, including memory stalls. At a clock rate of 600 MHz, more than 160 full-image wavelet transformations per second can be performed (grayscale images).

The cycle counts per pixel are also indicated in Table 6.7. It can be clearly seen that for higher resolutions (towards Level 0), the transform becomes more efficient, requiring less cycles per pixel. This is due to the fact that less memory stalls occur and boundary extensions occupy a smaller part of the image. For even higher resolutions, it is likely that this number will converge, as these two effects become insignificant.

Table 6.7: Obtained cycle counts for the 4-level 5/3 wavelet transform of a 4CIF image, executed at 600 MHz. Cycle counts are averaged over 1,000 frames. Percentages indicate the fraction of computations contained at that level.

Level	Mcycles	Percentage	Level size	Cycles/pixel
0	2.554	70.0%	704×576	6.30
1	0.695	19.1%	352×288	6.86
2	0.248	6.8%	176×144	9.80
3	0.149	4.1%	88×72	23.6
Total	3.647	100.0%	704×576	9.00

When applying the wavelet transform as part of the complete SVC, Table 6.8 gives the cycle counts for the transform for all color channels. The indicated percentages represent the cycle contributions as part of the single-frame encoding process.

Table 6.8: Computational complexity of the 5/3 wavelet per frame, for a 4CIF surveillance sequence. Cycle counts are averaged over 1,000 frames and the 'part' column indicates the percentage contribution as part of the whole SVC for one frame.

	Y channel		C_b channel		C_r channel	
Function	Mcycles	Part	Mcycles	Part	Mcycles	Part
Wavelet	3.648	12.41%	1.076	3.66%	1.083	3.68%

Two-Stage SPECK (TSSP)

The TSSP coding algorithm has been implemented on the earlier adopted DM642 DSP, as presented in Section 6.4, while the special handling of the most negative value representable by the two's complement representation is discussed in Appendix C. During the experiments and the associated measurements, we have observed that the DMA transfers are so effective, that image coefficients are always available in Level-2 cache for the algorithm access. This implies that measured calculation times are larger than memory transfer times. Table 6.9 shows the measured cycle counts for the frame-based wavelet coefficient encoding, including all memory transfers. The 'part' column indicates the contribution as part of the whole SVC for one frame, which consumes 29.398 Mcycles per frame.

Table 6.9: Measured computational complexity of TSSP per frame, for a 4CIF surveillance sequence. Measured cycle counts are averaged over 1,000 frames and the part column indicates the contribution as part of the whole SVC for one frame.

	Y channel		C_b channel		C_r channel	
Function	Mcycles	Part	Mcycles	Part	Mcycles	Part
TSSP Stage 1	1.531	5.21%	0.402	1.37%	0.402	1.37%
TSSP Stage 2	3.921	13.34%	0.740	2.52%	0.828	2.82%
TSSP Parser	0.037	0.13%	0.011	0.04%	0.012	0.04%
Total	5.489	18.68%	1.153	3.93%	1.242	4.23%

The measured Mcycle counts in Table 6.9 reveal several performance aspects. First, it can be observed that Stage 2 of TSSP requires approximately 2.5× the number of cycles for processing the luminance channel (Y) and 2× the cycles for processing the color channels (C_b and C_r). Even though Stage 2 processes only a small part of the coefficients, it is more complex because of the additional computations required for generating the bitstream for all bit planes. On the other hand, Stage 1 processing is simpler due to the group-wise processing of coefficients in the leafs

of the coding tree. Furthermore, it can also be seen that the TSSP parser is very fast and consumes less than 1% of the cycles with respect to TSSP processing stages. The TSSP parser consumes about 0.2% as part of the complete codec.

Evaluation of the TSSP memory transfer model

The above test setup for measurements only provides cycle count measurements based on internal hardware counters, expressed in Mcycles. As such, the cycle counts provide no information regarding the usage of these cycles. The cycle counts represent a mixture of the following four categories: (1) computations alone, (2) computations in parallel with DMA memory transfers (3) wait cycles for completing the DMA transfers and (4) memory transfers using the CPU. Because a large part of the algorithmic design and mapping effort has been spent on optimizing the parallelization of computations and background memory transfers, in the following an estimation is made how this parallelism in the memory transfers works out as part of the measured performance. This estimate is based on the theoretical bounds of the memory-transfer model derived in Section 6.4, for which the theoretical cycle counts are given in Table 6.10. The measured cycle counts of Table 6.9 are combined with the theoretical cycle counts of Table 6.10, which leads to the estimated distribution of processor cycles in TSSP given in Table 6.11.

Table 6.10: Processor cycles estimated for memory transfers for TSSP on a DSP running at 600 MHz based on the memory-transfer model derived in Section 6.4. Numbers do not always add up due to rounding.

Function	DMA memory transfers in parallel to computations [Mcycles]	CPU based memory transfers [Mcycles]	Total [Mcycles]
Stage 1	0.220	0.112	0.332
Stage 2	0.022	0.043	0.065
Total	0.242	0.156	0.397

Table 6.11: Estimation of the distribution of the processor cycles in TSSP, based on system measurements and the memory-transfer model.

Function	Computations alone	Computations & DMA transfers	CPU memory transfers	CPU wait cycles for DMA
TSSP Stage 1	78.4%	14.4%	7.2%	0.0%
TSSP Stage 2	98.3%	0.6%	1.1%	0.0%
TSSP Parser	0.0%	0.0%	100%	0.0%

The conclusions from this table are that no significant cycle counts are spent on waiting for DMA memory transfers. The TSSP parser performs no calculations and reorganizes the bitstream in memory using the CPU only. Furthermore, from the numbers it can be concluded that the optimizations in Stage 1 facilitate high parallelism for computations and memory transfers, while that Stage 2 consists mostly of computations. It should be noted that these percentages are based on the fastest possible memory transfers, which in real-life scenarios will not always occur. As a result, it is likely that a larger percentage of cycles is spent on CPU memory transfers and/or computations concurrent to DMA memory transfers.

Temporal filtering

When this TSSP coding implementation is integrated in the complete SVC system, the system performs overall better. On the average, the measured processing per frame takes 29.398 Mcycles, yielding a throughput rate of 20 fps. This improvement is due to the decorrelation of the input frames in the temporal filtering. The resulting difference frames contain significantly less information and as a result, TSSP can skip large portions of insignificant coefficients. The obtained complexity reduction in the entropy coding is therefore larger than the complexity increase of the temporal filtering.

Table 6.12: Measured computational power for the complete SVC specified by function, for a 4CIF surveillance sequence. Cycle counts are averaged over 1,000 frames.

Function	Mcycles	Percentage
Input conversion	1.729	5.88%
Temporal noise reduction	1.757	5.98%
BDLD temporal filtering	10.971	37.32%
5/3 wavelet	5.807	19.75%
TSSP encoder	7.884	26.82%
Other	1.250	4.25%
Total	29.398	100.00%

From Table 6.12, it can be observed that the computational complexity of the three coding stages (5/3 wavelet, TSSP encoder and BDLD temporal filtering), have the same order of complexity. The wavelet transform has straightforward processing and is mapped in a highly optimized way. When comparing the much more complex TSSP entropy coder with the wavelet transformation, it utilizes only a factor of 1.4 more cycles. This is partly explained by that fact that the difference

frames are already decorrelated and therefore can be encoded very efficiently, but it also shows the success of the architecture-aware design of TSSP, that enables a very efficient mapping and efficient use of SIMD and DMA.

Even though no motion estimation and compensation is included, the BDLD temporal filtering is computationally expensive, mostly due to the large amount of data that is processed. The absence of motion estimation does not lead to significant artifacts, but does increase the bit rate in case of motion more than with motion estimation and compensation. This is explained by the fact that difference frames are encoded with a certain BPR and are therefore always created at a certain quality, with and without motion compensation. Without motion compensation the frame difference is larger, which translates to a higher bit rate. This has also been verified visually.

Due to limitations of the hardware architecture, the original raw input images can neither be stored nor transmitted, due to the large data volumes involved. It is therefore not possible to provide numerical quality indications for this implementation, but the results from Chapters 3, 4 and 5 are applicable. In Figure 3.20, it was shown the BPR provides a quality control for the TSSP entropy coder, regardless of the considered sequence. This also applies when TSSP is implemented as part of the complete SVC. Based on the City and Crew sequences, it is expected to achieve between 35 and 37 dB PSNR at $BPR = 3$.

6.7 Conclusions

This chapter considers the mapping of the proposed SVC coding algorithm on a custom-designed surveillance camera and video encoder and its embedded computing platform. Starting from the cost, size and power consumption requirements, a programmable DSP is adopted as the preferred computing platform. The developed video coding system of the previous chapters, except HPPS, is fully mapped and implemented onto the computing platform. The mapping reported in this chapter involves the key processing stages of the SVC framework: (1) the spatial wavelet transform, (2) the TSSP wavelet coefficient encoding algorithm and (3) the temporal filter structure.

- *Spatial wavelet transform mapping.* SIMD and DMA mapping techniques are exploited, combined with an elegant memory usage, to support a multi-level transformation. The results are that the proposed process of background DMA transfers becomes so effective, that the ALUs are almost never starved for data input. Besides this, the mapping can execute a 4-level transform at 4CIF broadcast resolution in 3.65 Mcycles, including memory stalls. At a

clock rate of 600 MHz, this translates to a processing time of 6.08 ms, thereby satisfying the performance requirements for a real-time image/video encoding system. Our realization is a clear improvement, when compared to the implementation by Choi *et al.* [113], who reported a processing time of 13.25 ms for a 25% smaller image (VGA).

- *Two-Stage SPECK (TSSP).* This involves the mapping of the proposed TSSP wavelet coefficient encoder. The mapping of the two stages are individually optimized. The first, data-independent stage calculates the Significance Level (SL) of all nodes in the quadtree and stores this information in the SL buffer. The SL buffer is then used in the second stage to make coding decisions, where large insignificant parts of the quadtree are skipped. Furthermore, wavelet coefficients are encoded in parallel for all bit planes. As above, SIMD and DMA techniques and a self-managed Level-2 cache are utilized to enhance the efficiency. Of the two stages, Stage 1 is optimized most efficiently, due to its fixed access patterns and data-independent processing. The complete implementation of a TSSP-encoded video frame, results in a cycle count of 7.884 Mcyles for a 4CIF 4:2:0 image, which leads to a throughput rate of 75 TSSP encoding cycles per second at 600 MHz.

- *BDLD temporal filtering structure.* This part is implemented without motion estimation and compensation, due to architecture limitations. By including the full BDLD temporal filtering structure, the overall efficiency of the execution of the complete SVC framework is increased, due to the coding of differential images instead of normal images.

For the execution of the whole SVC framework, it has been measured that the processing per frame consumes 29.398 Mcycles on the average, yielding a throughput rate of 20 frames per second. This throughput rate of the overall framework indicates and validates our claim for real-time performance of the proposed t+2D SVC framework. However, this conclusion comes with a remark that the motion estimation has been omitted. Since the developed HPPS ME algorithm was designed for efficient computation and memory bandwidth, we are confident that adding this algorithm will lead to a result that has the same order of magnitude in throughput rate. This confidence is based on the consideration that when the HPPS algorithm is added, the overall execution would still be based on the coding of difference images.

Reflecting on the algorithmic design and an embedded system mapping, we have found that the consideration of computing architectures during algorithmic design is very fruitful and should be considered as an aspect for an engineer de-

signing real-time imaging systems. It has been shown how critical changes to the algorithm, such as splitting the entropy coding in a data-independent and data-dependent stage, can bring significant benefits to the mapping, while retaining coding performance.

CHAPTER 

Conclusions

One of the greatest discoveries a man makes,
one of his great surprises,
is to find he can do what he was afraid he couldn't do.

HENRY FORD

Abstract

This thesis has presented the algorithmic design of a complete SVC coding algorithm, with a focus on video surveillance and embedded systems. We have made proposals for optimized algorithms with respect to mapping complexity in the fields of wavelet coefficient entropy coding, temporal configurations and motion estimation. In the previous chapter, we have mapped this SVC to an embedded system to validate the mapping complexity. In the concluding chapter of this thesis, the contributions of the individual chapters are summarized, and provide answers to the research questions posed in Chapter 1. Finally, we will reflect on the key and open issues and provide a future perspective for scalable video coding and video surveillance.

7.1 Conclusions of the individual chapters

Chapter 2: Developments in wavelet image and video coding.
This chapter has introduced the reader to Scalable Video Coding through explaining the need for video compression along with a short history of current image and video coding. It then focusses on wavelet-based scalable coding, first by introducing common wavelet transforms and more specifically integer wavelet transforms based on the lifting implementation. The multi-level 2D dyadic wavelet decomposition was presented, along with 1D, 2D and multi-level energy correction algorithms. Finally, the chapter provides the reader with a very detailed description of two state-of-the art wavelet coefficient encoding algorithms: SPIHT (Set Partitioning in Hierarchical Trees) and SPECK (Set Partition Embedded bloCK). The detailed description illustrates their inner working, with detailed encoding and decoding examples.

Chapter 3: Hardware-efficient scalable wavelet coefficient coding.
This chapter explores the algorithms for encoding wavelet coefficients specifically for image coding. First the complexities of the SPIHT and SPECK coding algorithms are investigated, after which we have proposed a hardware-efficient scalable image coder based on SPECK, called TSSP (Two-Stage SPECK), that is backwards compatible to SPECK. We have proposed several enhancements that significantly improve the algorithm's efficiency. First, processing is split into one data-independent stage and one data-dependent stage. A highly efficient buffer eliminates the need for dynamic lists, and processing in the second stage is performed for all bit planes in parallel. An enhancement of the algorithm is the Highly Scalable (HS) extension, which allows parsing of the bitstream without payload decoding, thereby creating a bitstream of any desired quality, resolution and bitstream order. The second enhancement, 5/3 Energy Correction (53EC), retains the perfect reconstruction property of the 5/3 integer wavelet, while significantly improving lossy coding performance by up to 5 dB, with a negligible performance drop of only 0.1% at lossless coding.

Chapter 4: Complexity in the temporal domain of scalable video coding.
This chapter presents the trade-off between the complexity and performance of several suitable coding architectures: predictive coding, multi-layer scalable predictive coding and 3D subband techniques: t+2D, 2D+t and 2D+t+2D. Based on trading-off complexity, coding performance, scalability and domain-relevant features, we have adopted the t+2D configuration. For this configuration, a framework is presented to evaluate various temporal configurations, of which we have adopted a structure called BiDirectional Low Delay (BDLD). This configuration

features low memory access and low end-to-end delay, while still achieving sufficiently high coding performance. To improve computational complexity and video quality, two extensions to the t+2D framework have been proposed. The first improves energy correction in the temporal lifting tree and significantly reduces computational complexity and required memory bandwidth. The second improves the average quality of the frames within a GOP and reduces quality fluctuations significantly, which is also expressed by a reduction of the variance of the PSNR by 30% and a significantly improved perceptual quality. In all, the coding quality approaches that of H.264 SVC within 1 dB.

Chapter 5: Motion estimation for scalable video coding.
This chapter has presented Motion Estimation (ME) from the perspective of Scalable Video Coding (SVC) and accelerators from mediaprocessor architectures. SVC requires that the ME algorithm has to operate with frames at varying temporal distances, so that new methods of motion-vector propagation are necessary. Based on this aspect, a novel ME algorithm has been proposed, named Highly Parallel Predictive Search (HPPS). HPPS differs from previous proposals in that it utilizes three independent processing paths: one for the $(0,0)$ vector, one using a spatial predictor, and one using a temporal predictor. The two predictive candidate vectors are surrounded with additional refinement candidates arranged in a dense PSS pattern, which reduces SAD test points up to 50% without significantly degrading the accuracy of the found motion vectors. The performance of EPZS is considered state-of-the-art and HPPS has a similar performance with respect to the measured rate-distortion curve. The main benefit of HPPS lies in its characteristic features with respect to real-time implementation. First, HPPS has a fixed computational load, regardless of scene activity and the temporal distance between frames. Second, HPPS is very predictable in its data usage. EPZS and HPPS both approach the full-search reference within 0.05 dB for SD sequences and 0.03 dB for HD sequences.

Chapter 6: Real-time algorithmic validation on an embedded architecture.
This chapter presents optimized implementations of 2D wavelet filtering, the TSSP wavelet coefficient encoder and the temporal filtering framework to validate execution on an embedded DSP platform within the framework of a custom-designed video surveillance camera. We have achieved efficient implementations by using SIMD (Single Instruction Multiple Data) instructions for data and computational parallelism, and Direct Memory Access (DMA) to transfer data to and from our self-managed Level-2 cache, parallel to computations. The performance of the implemented framework based on executing a 4-level wavelet transform at 4CIF (CCIR-601) broadcast resolution, results in an execution time of 6.08 ms, while for

TSSP, real-time performance is obtained. The full implementation includes temporal filtering, which also yields coding of differential images. This fully scalable video encoding system obtains a throughput rate of 20 fps, however, without the integration of the proposed HPPS ME algorithm.

7.2 Discussion on research questions and contributions

This section returns to the research questions posed at the beginning of this thesis in Section 1.4, supported by the findings of this thesis.

RQ1: Efficient Variable Length Coding (VLC) of scalable transformed image and video data.

RQ1a: *Is it possible to design an alternative high-performance scalable video coefficient coding algorithm without losing coding fidelity?*

RQ1b: *Can we extend beyond the scalable properties of current scalable codecs, and provide highly flexible coding, with only a limited increase of codec complexity?*

RQ1c: *Is it better to implement scalable variable length coding in a quality-progressive way, or to process multiple qualities in parallel?*

All of the above research questions are answered using Chapter 3 as a reference. In Section 3.2 we have proposed the Two-Stage SPECK (TSSP) wavelet coefficient codec, in which several modifications have been made to the SPECK codec. These modifications significantly improve its computational performance by splitting the coding into two stages. The first, data-independent processing stage calculates significance information for the whole spatial tree and this stage can be highly optimized for implementation. As a consequence, this allows the second, data-dependent stage to make coding decisions without observing individual coefficients, and large regions can be skipped altogether if they are insignificant. These optimizations answer the first part of research question RQ1a. Furthermore, TSSP creates a bitstream identical to the state-of-the-art SPECK codec, which is well-known for its coding fidelity, which answers the second part of research question RQ1a.

Additionally, two extensions have been proposed to the novel TSSP codec (Section 3.4), which give answer to research question RQ1b. The first extension is a Highly-Scalable (HS) mode, where data is organized in data blocks for every combination of each resolution in the dyadic composition and each bit plane. Header information is included to allow for fast access to these data blocks, and stuffing bits provide data alignment of data blocks with the data processing structures in

hardware and software. The second extension expands this HS mode for 5/3 integer wavelets, which allows it to be used for both lossless and lossy coding without compromise. This is accomplished by implementing separate data blocks for the LL, joined LH–HL, and HH wavelet frequency bands and adjusting the amount of quantization for each of these bands, thereby effectively implementing energy correction that ensures smooth lossy coding. Both extensions add a low overhead cost (header data and stuffing bits).

With respect to RQ1c, we have found that it is better to process multiple quality levels in parallel, even if the desired output bitstream needs to be created in a quality-progressive way. By avoiding the necessity for constant rate checking, and creating a single access for each coefficient only, the TSSP codec creates a full-quality bitstream in nearly the same time as it would take to process a single quality level (bit plane). Due to the HS scalability mode, the scalability of the codec is fully exploited by creating a quality-progressive bitstream, which is implemented through selective reordering of data blocks.

RQ2: Framework for and complexity estimation of scalable temporal video coding.

RQ2a: *Can we identify a suitable framework for scalable temporal coding?*
RQ2b: *What is the trade-off between temporal coding complexity, end-to-end coding delay and corresponding visual quality?*
RQ2c: *Based on the outcomes of the previous trade-offs, is the chosen temporal coding structure suited for video surveillance?*

Regarding research question RQ2a, in Chapter 4 we investigated several coding architectures, both predictive (conventional and scalable) and 3D subband coders (t+2D, 2D+t and 2D+t+2D). For each coding architecture, the following performance metrics were addressed: inherent and motion complexity, coding performance, scalability in quality, resolution and frame rate, and surveillance domain-specific features such as end-to-end delay, trick play and random access. Based on this evaluation, we have selected the t+2D framework for its inherent low complexity, good scalability options for resolution and quality. The inherent benefit of the t+2D architecture is that motion estimation and compensation can be performed in the time domain so that the efficiency of the total coding system should be comparable to the well-known H.264 coding standard. Besides this, it is known that wavelet coding is slightly more efficient than DCT coding. These arguments confirm that the proposed framework is suitable for scalable temporal coding.

To investigate the trade-off between temporal coding complexity, end-to-end coding delay and corresponding visual quality (research question RQ2b), a framework was proposed with four basic building blocks to create arbitrary temporal configurations for the t+2D coding framework. These building blocks were utilized to create several deliberately constructed temporal configurations. By calculating the Rate-Distortion (RD) performance, we observed that the proposed configurations resulted in smooth curves similar to other state-of-the-art video coding algorithms, even though the internal structure of the proposed coding algorithm is significantly different. It was found that the suitable operation interval where this smooth behavior occurs, is within 1 dB of other well-known existing coding standards. As expected, more complex temporal configurations lead to improvements in rate-distortion performance, but also in an increase in end-to-end coding delay.

Regarding research question RQ2c, a suitable temporal configuration was found by elegantly managing frame dependencies, so that the best quality is achieved with an acceptable end-to-end delay (3 frames). This BiDirectional Low Delay (BDLD) configuration has a moderate complexity (27 ME actions per GOP of 16 frames), and a workable frame memory size of 8 frames. This leads to a video output of high quality, both visually and numerically (a 4CIF video at 30 fps yields a PSNR of 37 dB at 3 MBit/s). We furthermore proposed two extensions (TEC and LCEF) to the t+2D framework which achieve an even higher quality and/or a lower complexity. With respect to the obtained quality and complexity, we conclude that the proposed system is suitable for surveillance applications and offers integrated scalability in its coding framework. The disadvantage of the proposal is that it is not a standard at the moment, so that the integrated development cost is higher.

RQ3: Motion estimation in scalable video coding and its implementation efficiency.

RQ3a: *Can we develop a motion estimation algorithm that is suitable for a temporal scalable wavelet coding system and that elegantly propagates motion information within such a system?*

RQ3b: *Is it possible to design a motion estimation algorithm that is not only efficient, but also based on the hardware media processing kernels of modern computing platforms?*

In Chapter 5 a Motion Estimation (ME) algorithm has been developed, called HPPS, that is based on two motion-vector predictors, around which further candidate vector positions are tested in a dense search pattern. The HPPS ME algorithm

facilitates temporal scalable video coding through providing: (1) robustness for uncertainties in the temporal predictor through the dense search, (2) a fixed computational complexity, regardless of the time difference between frames and (3) a performance similar to that of state-of-the-art algorithms which have a sequential nature. The proposed ME algorithm approaches the performance of the full-search reference ME. Good results for propagation have been obtained for both the proposed HPPS ME algorithm and the state-of-the-art EPZS ME algorithm. This propagation is created by grouping a set of co-located motion vectors into one representative vector for a bidirectional pair of motion estimations, utilizing coding mode decisions made by the temporal coding algorithm. This representative vector can then be easily propagated within the same temporal hierarchical level and between these levels.

The answer on the second research question (RQ3b) involves several aspects. The reuse of block matching allows HPPS to fully utilize block-based hardware accelerators, and the dense search is fully implemented with block SAD and block rotation accelerators. This dense search combined with only 2 predictors, reduces bandwidth requirements and results in fast testing of candidate motion vectors, and allows for pre-fetching of data blocks using Direct Memory Access (DMA).

RQ4: Verification of scalable VLC and scalable temporal coding on embedded architectures.

RQ4a: *What embedded execution architecture would be suitable for implementing a scalable video coding algorithm?*

RQ4b: *How can a combination of the high-performance scalable video coefficient coding and the scalable temporal coding be mapped on a resource-constrained embedded architecture, and can real-time performance be obtained?*

A custom-designed video surveillance camera and its embedded computing platform was presented in Section 6.2, which answers research question RQ4a. This camera and platform are chosen based on cost, size and power consumption requirements. For efficient execution of the processing, a programmable DSP has been adopted as the preferred computing platform. This DSP provides hardware accelerators such as SIMD and DMA that facilitate efficient mapping of algorithms. Nowadays, state-of-the-art DSPs offer sufficient parallelism over multiple cores and contain accelerators for multi-media coding and processing. The clock frequency of such a DSP is also high enough for obtaining real-time performance.

The utilization of the SIMD and DMA hardware accelerators for the wavelet transform (Section 6.3), TSSP (Section 6.4) and the temporal coding (Section 6.5),

provide the basis for mapping the complete coding system. For the wavelet transformation, the process of background DMA transfers is so effective, that the ALUs in the DSP are almost never starved for data input. The TSSP coding can also be efficiently executed when using background DMA transfers and heavily exploiting SIMD instructions. The following statements hold for real-time performance. A 4-level wavelet transform at 4CIF broadcast resolution requires a processing time of 6.08 ms, which is only a fraction of the required 30-40 ms frame time. The complete TSSP encoded process requires 7.884 Mcyles for a 4CIF 4:2:0 image, which leads to a throughput rate of 75 TSSP encoding cycles per second. By adding the temporal coding structure, overall coding efficiency is actually improved. The combination of all three native blocks of the coding system yields a throughput rate of 20 fps. Although the motion estimation is missing in this mapping, it is known the ME algorithm complexity is low and fits to the DSP architecture. Therefore we conclude that the coding system can indeed achieve real-time performance when some more development time would be invested.

7.3 Key issues and open topics

Motion estimation complexity contributes significantly to the overall complexity of any video coding algorithm. Due to unforeseen problems with the tooling of a next generation hardware architecture, we were unable to validate the HPPS motion estimator. This next generation hardware architecture involved two cores, one special programmable ALU implemented in hardware logic and a co-processor featuring advanced multi-media instructions. Unfortunately this specific architecture did not offer sufficient programming tools to map the HPPS algorithm smoothly. As a result, the main purpose of the redesign of the ME algorithm, shifting the focus of optimization from the SAD calculation to memory bandwidth, could not be validated. Therefore, in future research, an alternative architecture should be chosen that includes the proposed hardware accelerators, so that a proper analysis can be made.

With current trends in *multi-core processing*, new opportunities and challenges arise. While the utilized DSP does exploit advanced data and computing parallelism, its architecture is still elementary different from the present multi-core architectures gaining popularity. The DSP contains multiple cores within its ALU with shared caches, while modern multi-core architectures feature more independent processing cores. These independent processing cores can be fully exploited when the processing is performed within local areas of the data fields (data locality). Let us provide an example how this parallelism can be provided with the current system. For TSSP, processing in Stage 1 and 2 can be easily split to facilitate

parallel implementations. For example, the image can be split in four quadrants, where all quadrants are processed in parallel. In Stage 1 of TSSP, the SL buffer can be calculated in parallel using this strategy by assigning separate computing instantiations to the four individual independent image regions, resulting from the quadtree partitioning. When all four regions are completed, the first value of each of the four regional buffers is used to calculate the SL of the combined block, and this value and the four regional buffers are cascaded to form the merged SL buffer. Parallel processing can be implemented at any desired level of the quadtree, and therefore processing can be split in 4, 16, 64, etc. blocks. Parallelization in Stage 2 works in a similar manner, using the division of computation at any desired level of the quadtree.

7.4 Future outlook on scalable video coding

Standards vs. custom design

When designing a custom video coding system, a comparison with the current video coding standards is inevitable. The research community for H.264 and beyond is so large, with backing from companies worldwide, that it is nearly impossible to successfully take a different path. In a commercial setting, this constraint is even more compelling, because many cost-effective hardware solutions for H.264 are available on the market, due to economies of scale involved. However, the H.264 standard is developed primarily for systems that support a limited number of streams. Most current video decoders can only decode a maximum of two streams, and software solutions on general-purpose CPUs even have difficulty decoding a single high-resolution stream. For applications where the requirements significantly differ from the mainstream applications, it is possible to compete with H.264.

Key applications and requirement differentiation

Surveillance applications could become a key application for scalable video coding because the amount of implemented codecs is growing in a rapid pace due to an ever growing installed base of surveillance cameras. Although scalable video coding may be an attractive choice for surveillance applications, such coding systems are not widely adopted elsewhere. Even though the SVC extension exists for H.264, it is hardly used in practice.

Furthermore, for surveillance applications, there is a large differentiation in requirements when compared to video standards driven by consumer requirements. In the video surveillance domain, it is critical to do real-time encoding on a low-cost system, while simultaneously supporting the decoding of hundreds of streams at a single location. The lack of market momentum for H.264 SVC, and

the clear requirement differentiation of surveillance applications, provides a niche segment in which a custom-designed system is feasible, so that it can be tailored to the specific requirements of the application area, as discussed in this thesis.

Future vision: object-based coding and content analysis

Beyond the applications described in this thesis, the concept of scalable coding paves the way for novel applications of this technology, such as object-based coding. Experiments have been conducted to investigate object-based coding, in which the wavelet transform was adapted to perform symmetric filtering at object boundaries. With such modifications, objects can be filtered and encoded individually. Shape information can also be encoded using wavelets, and when applying a novel transform, shape information can be encoded using the TSSP encoder. This novel transform utilizes binary inputs, and the low-pass filter is defined as the binary 'OR' operator. This is essential for proper reconstruction and encoding of DC information.

Furthermore, there is a significant growth in automatic processing of surveillance video through content analysis. Rule-based systems are already in operation, where for example the presence of an object in a certain perimeter triggers an action, such as an alarm. Behavioral analysis is also becoming increasingly popular, such as the tracking and trend analysis of the movement of persons or objects. One category is out-of-the-ordinary movement detection, such as left-luggage detection, ghost drivers on a freeway, running people in a crowd, or an elderly person falling in a treatment home. Multi-camera content analysis expands the behavioral analysis from a single room, to the complete premises under surveillance, allowing for advanced analysis.

When scalable video coding, object-based coding and content analysis are combined, a variety of interesting applications are enabled. Once a separation is made between foreground and background objects in a video surveillance scene, the more important foreground objects can be encoded at a higher quality, while the background objects are encoded using a best-effort strategy. When combining this with a network supporting Quality of Service (QoS), it is possible to assign the data from the foreground packets with a higher priority. In case of network congestions, the data for the background objects will be rejected, and foreground objects are still transmitted. When the QoS scheme provides many layers of granularity, each particular bit plane, resolution and temporal block of the highly scalable TSSP bitstream can be given different priorities, and a very graceful degradation scheme can be implemented.

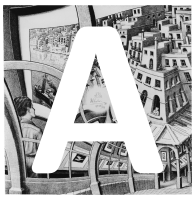
This would for example provide robustness for systems that perform face and license plate recognition. A two-stage content analysis approach can be proposed,

with object-based scalable video coding as a support layer. In the camera, an initial detection of objects of interest is performed, but without classification (e.g. the location of a license plate, or a face). Such objects of interest are then coded at a higher quality (foreground), while the remainder is coded at a lower quality (background) to provide context. At another location, specialized license plate and face recognition algorithms can then be employed, in real-time or as post-processing. By utilizing the highest quality images for the regions of interest, the accuracy of these algorithms is directly improved.

With the advancements in image and video editing software, content can be manipulated, while being visually indistinguishable from actually recorded scenes. This phenomenon is clearly visible in our society through numerous viral videos that have appeared on the Internet, which are indistinguishable from real videos. In video surveillance, it is important that recorded videos have not been tampered with, as they can be used as evidence in a court of law. Closed-circuit systems can facilitate this by creating encoders that allow verification of the video through, for example, an authentication scheme using private and public keys, similar to email. Even though surveillance systems commonly do not provide this authentication, it is the opinion of the author that this ability of authentication will become crucial in the near future.

Appendices

APPENDIX



TSSP and wavelet modifications for reduced complexity

In this appendix, we investigate the complexity reductions we can obtain by adjusting the location of quality control in TSSP. We propose two different modifications to obtain a lower complexity by performing bit-plane dropping in either the wavelet transformation (Section A.1), or in Stage 1 of the TSSP (Section A.2), instead of the regular bit-plane dropping in Stage 2 of TSSP described in Chapter 3.

For both modifications, the TSSP bitstream remains intact, and therefore the TSSP decoder and inverse wavelet transform do not have to be modified and the rate-distortion performance is not affected. The number of bit planes dropped is controlled by the Bit-Plane Reduction parameter BPR.

A.1 Quality control in the integer wavelet

To reduce complexity in Stage 2, we propose to include the bit-plane dropping fully within the wavelet transformation. The 5/3 integer wavelet filter is modified to include the quality control by dropping the lower bit planes during the de-interleaving of the wavelet coefficients. For all but the last 2D wavelet transform in the multi-level dyadic decomposition from Figure 2.2, the coefficients of the HL, LH and HH bands are modified to include the quality control, as the LL band will be processed further. For the last iteration of the 2D wavelet transform, bit-plane dropping for the coefficients in all four bands is performed.

Unfortunately, it is not possible to simply bit-shift the wavelet coefficients to the right by BPR, as this will create an error for negative values, such as the value -1. To match the behavior of the TSSP codec, the sign is included to correct for the behavior of the bit-shift operator. The correct dropping of bit planes is achieved through Equation (A.1), which is specified as

$$coef = \left\lfloor \frac{coef + (2^{BPR} - 1) * (1 - \text{sign}(coef))}{2^{BPR}} \right\rfloor, \tag{A.1}$$

with $coef$ and BPR as defined earlier, and $\text{sign}(x)$ denotes the sign as a unit-step function for positive values. The floor operator results from the use of the shift-right operator in fixed-point arithmetic to implement the division by 2^{BPR}.

To examine the working of Equation (A.1), let us elaborate with the numbers 7,8,9 and their negative counterparts -7, -8 and -9. In two's complement binary notation these numbers are represented as:

For the positive numbers, it is easy to see that when dropping 3 bit planes ($BPR = 3$), the resulting value of Equation (A.1) is 0, 1 and 1 for 7,8 and 9, respectively, easily accomplished by the shift right operator in binary arithmetic. Negative numbers should show similar behavior, and result in 0, -1 and -1. If we use the shift right operator to implement the bit-plane dropping, it would yield

Number	Two's complement binary notation	With $BPR = 3$
7	00000111	0
8	00001000	1
9	00001001	1
-7	11111001	0
-8	11111000	-1
-9	11110111	-1

wrong results, leading to -1, -1 and -2 for -7, -8, and -9, respectively, since applying the shift right operator to negative values will always result in a negative number (due to the two's complement representation).

To accomplish proper bit-plane dropping in fixed-point arithmetic and two's complement values, we have implemented Equation (A.1) as

$$coef = (coef + ((1 \ll BPR) - 1) * (coef < 0)) \gg BPR, \tag{A.2}$$

with $\ll$ and $\gg$ denoting shift left and right operators and < 0 acts as a sign determination.

Using this formula, the output is as desired. This is also true for Equation (A.1), but the floor operator on negative numbers may cause some confusion. For the numbers -7, -8 and -9 we obtain:

$$\left\lfloor \frac{-7 + (8-1) * (1-0)}{8} \right\rfloor = \left\lfloor \frac{-7+7}{8} \right\rfloor = \lfloor 0 \rfloor = 0, \tag{A.3}$$

$$\left\lfloor \frac{-8+7}{8} \right\rfloor = \lfloor -0.125 \rfloor = -1, \tag{A.4}$$

$$\left\lfloor \frac{-9+7}{8} \right\rfloor = \lfloor -0.25 \rfloor = -1. \tag{A.5}$$

For $BPR = 0$, the output matches the original wavelet, which was discussed in Section 2.3. With $BPR > 0$, the wavelet transform effectively performs the new quality control by dropping BPR bit planes. As a result, the output of this modified wavelet transform has significantly lower wavelet coefficients, which if implemented properly, also reduces storage requirements and memory bandwidth. Furthermore, Equation (A.1) can be combined with energy correction for LL and HH bands in the 5/3 wavelet [7] [31].

Stage 1 of the TSSP is left unmodified, and Stage 2 of the TSSP is simplified by omitting the BPR significance check for both blocks and individual coefficients.

To create a standard TSSP bitstream, the header information is corrected with the BPR parameter.

A.2 Quality control in TSSP Stage 1

At Stage 1 of the TSSP, the SL buffer is created from the wavelet coefficients as discussed in Section 3.3. The dropping of bit planes is achieved by adjusting the calculation that determines the SL of the wavelet coefficients. In the original TSSP, the SL is calculated by Equation (A.6), and defined by

$$SL = \text{lmb}(\text{abs}(coef)), \tag{A.6}$$

with $coef$ denoting the wavelet coefficient and $\text{abs}(x)$ stands for the absolute value of x. The $\text{lmb}(x)$ operator returns the position of the left-most bit, counted from the lsb to the msb, starting with one, and is zero for $x = 0$. To enable bit-plane dropping at Stage 1, Equation (A.6) is modified to include the BPR parameter to create Equation (A.7), which is defined by

$$SL_{BPR} = \text{lmb}(\left\lfloor \frac{\text{abs}(coef)}{2^{BPR}} \right\rfloor). \tag{A.7}$$

The floor operator results from using the shift-right operator in fixed-point arithmetic, which implements the division by 2^{BPR}. Equation (A.7) will result in a behavior similar to the bit-plane dropping in TSSP. For example, a coefficient value of 7 will have $SL = 3$ according to Equation (A.6). With $BPR = 3$, this coefficient would be considered insignificant, as $7 < 2^{BPR}$. The floor operator reduces $\frac{7}{8}$ to 0 in Equation (A.7), resulting in the correct $SL_{BPR} = 0$.

Stage 2 of the TSSP can now be simplified, as it can omit the significance check against the BPR parameter. Significance of blocks is then tested against zero, and during encoding of individual coefficients, this check is replaced with a right-shift of the wavelet coefficient by BPR.

A.3 Experimental results

In this section, we will show the complexity reductions obtained by moving quality control to other parts of the TSSP coding process. We implemented the two proposed complexity reducing modifications into the original TSSP and wavelet transform, alongside two regular methods of quality control, to measure the impact on processing time. Four different scenarios of Bit-Plane (BP) dropping are investigated: (1) lossless coding with BP dropping in the parser, (2) BP dropping

in Stage 2 of the TSSP, (3) BP dropping in Stage 1 of the TSSP, and (4) BP dropping in the wavelet transform.

The algorithms are implemented in C and the measurements were performed on images first utilized in Chapter 3. These images are listed in Table 3.1, and visualized in Figure3.13 at the beginning of Section 3.5. From zero to five bit planes are dropped for each of the images. The possibility of reduced word-lengths, which is possible with bit-plane dropping in the wavelet, is not investigated in the PC implementation.

Wavelet Complexity

For the wavelet, we have compared execution time with and without the quality control within the wavelet for various images in Table A.1. As expected, we observe that the execution time is not affected by the number of bit planes dropped. Furthermore, bit-plane dropping in the wavelet requires approximately 6.6% more processing time for the high-definition images, and 7.6% for standard-definition images, compared to the regular wavelet without bit-plane dropping.

Table A.1: Complexity increase of the wavelet with quality control compared to the regular wavelet for BPR 0, . . . , 5.

BPR	1920×1080	704×576	640×480	512×512
0	6.60%	7.66%	7.53%	7.52%
1	6.56%	7.64%	7.55%	7.61%
2	6.54%	7.66%	7.55%	7.61%
3	6.37%	7.65%	7.55%	7.66%
4	6.58%	7.65%	7.54%	7.64%
5	6.59%	7.66%	7.54%	7.68%

Entropy Coding Complexity Reduction

To estimate the computation complexity reduction of the proposed modifications, we measured the execution time for the four different scenarios of bit-plane dropping, each with 0 to 5 bit planes dropped. The measurements are visualized in Figure A.1, normalized to lossless encoding ($BPR = 0$) with bit-plane dropping in the parser, for the 1920×1080 pixel image Dude, which shows an average reduction of processing speed (complexity) among the five high-resolution images. Other images show similar reduction curves.

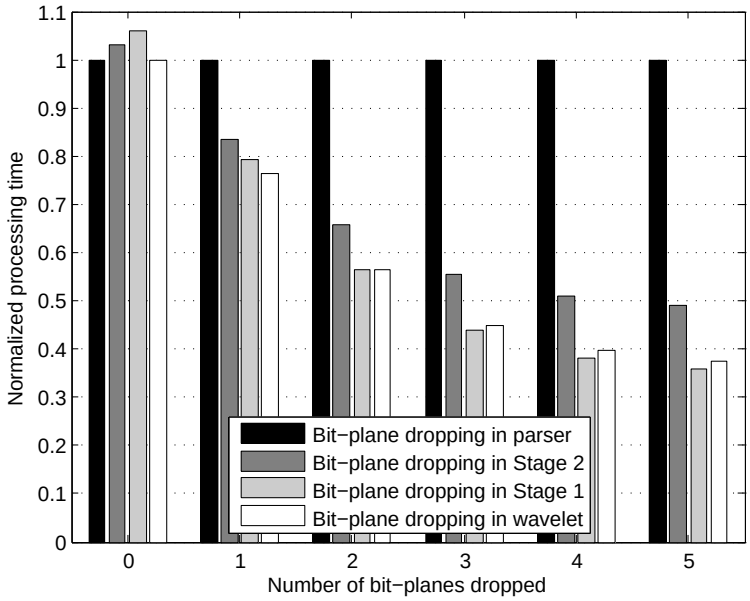


Figure A.1: TSSP normalized processing time for BPR $0, \ldots, 5$.

As expected, for $BPR = 0$, we observe that even though no bit planes are dropped, quality control within the TSSP at Stage 1 or 2 adds some complexity to the algorithm. Furthermore, for higher values of BPR, an overall decline in computation time is observed. This is due to the fact that Stage 2 of the TSSP is data-dependent, and performs less computations when bit planes are dropped, as larger parts of the quadtree are insignificant. This decline is not applicable when quality control is performed in the parser, after the TSSP, as all original coefficients are processed first without quality control.

Most importantly, we observe a complexity reduction when bit-plane dropping is performed in Stage 1 of the TSSP or the wavelet, compared to bit-plane dropping in Stage 2. It should be noted that while bit-plane dropping in the wavelet reduces complexity in the TSSP, the wavelet needs to be modified and requires 6.6% more processing time.

Table A.2 lists the processing time reduction of the TSSP when quality control is performed in Stage 1, compared to the original method of quality control in Stage 2, for all images. We observe a processing time reduction of the TSSP of up to 48%, and in the typical scenario of 1920×1080 images and a BPR of 3, we achieve an processing time reduction of 28%.

Table A.2: Processing time reduction of TSSP with quality control in Stage 1 compared to TSSP with quality control in Stage 2.

	BPR					
Image	0	1	2	3	4	5
Videoclip	-3%	3%	18%	34%	41%	44%
Bob Marley	-3%	3%	16%	26%	33%	39%
Dude	-3%	5%	16%	27%	34%	37%
Pipe	-3%	5%	14%	22%	28%	32%
Eye	-3%	7%	23%	31%	35%	37%
City still	-3%	2%	8%	16%	27%	39%
Crew still	-2%	3%	13%	25%	36%	42%
Space	0%	2%	7%	14%	23%	37%
Lena	-3%	2%	12%	27%	39%	48%

Figure A.2 presents the normalized processing time for the combination of the wavelet and the TSSP codec. Again the complexity reduction of performing quality control in Stage 1 and the wavelet is observed. For TSSP, when the complexity of the wavelet is also taken into account, bit-plane dropping in Stage 1 becomes more favourable than bit-plane dropping in the wavelet, as no modifications need to be made to the wavelet which increase its complexity.

Conclusion

We have proposed two complexity-reducing modifications to the quality control in bit-plane based coders, which retain the original bitstream and thus rate-distortion performance. The first generic modification, moves quality control to the wavelet transform, slightly increasing its complexity. For TSSP, we have also investigated moving quality control from Stage 2 to Stage 1, leaving the wavelet unaltered. Both methods show a significant and similar reduction in processing time. We found that for TSSP, bit-plane dropping in Stage 1 is preferred in terms of complexity and processing time, as it does not require any modifications to the wavelet transform. We observe an processing time reduction of the TSSP up to 48%, compared to the standard TSSP. In the scenario of 1920×1080 pixel images with a typical level of quality control (3 bit planes dropped), we achieve an average processing time reduction of 28%. Furthermore, we have found that moving quality control out of the entropy coding into the wavelet transformation is generically applicable to other bit-plane based codecs, while offering nearly the same processing speed.

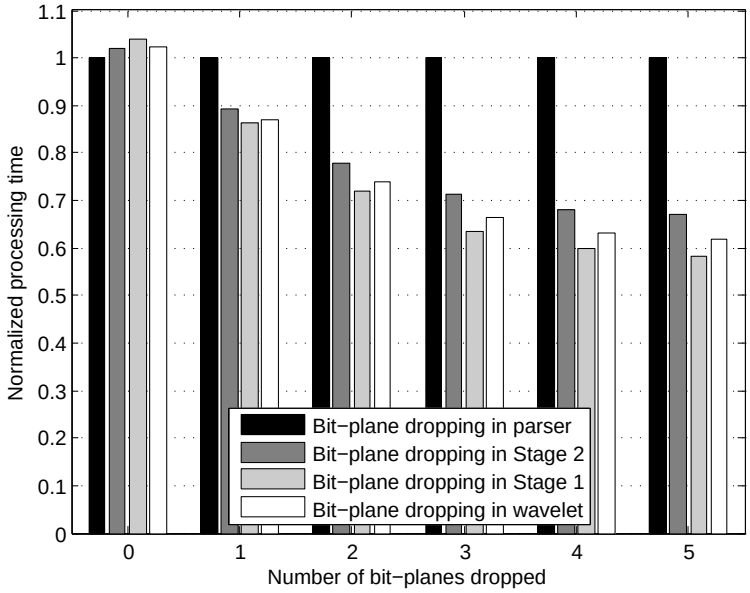
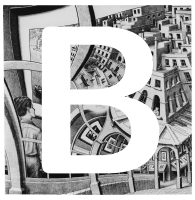


Figure A.2: Normalized processing time for the DWT and TSSP combined, for BPR 0, . . . , 5.

APPENDIX



The TSSP bitstream

In this appendix, the Highly Scalable (HS) bitstream definition of TSSP (Chapter 3) is given, including the 5/3 Energy Correction (53EC) extension.

B.1 Major stream components

The TSSP bitstream consists of two parts, a header and the entropy data. These two parts will be discussed in more details in the following sections, where the bitstream is always represented by greyed blocks. Tables will also be included to describe the variables in the bitstream and to specify their size.

TSSP bitstream:

B.2 Header information

The header always contains three parts and begins with image and stream information. After that two tables are included, the first with the minimum and maximum bit planes for each of the resolution layers. The second table indicates the number of sorting and refinement bits per resolution/bit-plane block. The two tables are slightly different for the TSSP and the TSSP53, which will be clarified in the respective sections.

Image and stream information

The image and stream information contains information required to decode the stream and decode the image.

Item	Size	Description
width	uint16	Width of the image
height	uint16	Height of the image
decomp	uint4	Number of wavelet decompositions
resLvl	uint4	Number of resolution levels
sigBits	uint4	Signalling bits
alignment	uint4	Stream alignment in $2^{alignment}$ bits, e.g. $alignment = 3$ gives a byte aligned stream

The four signalling bits are defined as:

reserved4	diagonal53	qtRounding	dataOrder

Item	Size	Description
reserved4	bit	Reserved for future use, set to 0
diagonal53	bit	Method of Bit-Plane Reduction (BPR): 0 = Flat BPR 1 = Diagonal BPR for 53 NoEC and TSSP53
qtRounding	bit	QuadTree Rounding: 0 = floor, 1 = ceiling
dataOrder	bit	Data order of stream: 0 = quality progressive, 1 = resolution progressive

The diagonal53 bit indicated whether the BPR is applied over blocks with the same bit plane (flat), or for different bit planes at different resolutions (diagonal). The flat BPR is used for the regular TSSP and the diagonal BPR for the TSSP53. This bit therefore also indicates the difference between a TSSP and a TSSP53 bitstream.

Minimum and maximum bit-plane table

For each resolution level, the minimum and maximum bit planes are put in the minimum and maximum bit-plane table. It starts at resolution level 0 (the smallest low-pass band), and then the following resolution levels (high-pass bands). To indicate no data is available in a particular resolution layer, the maximum is set to 0, and the minimum to 15. For the regular TSSP, one minimum-maximum pair is stored for each resolution, which is shown in the following table.

	res[0]	res[1]	...	res[n]
High Nibble	Max bitpl	Max bitpl	...	Max bitpl
Low Nibble	Min bitpl	Min bitpl	...	Min bitpl

For the TSSP53, a low and high value is available for each resolution, for both the joined LH–HL band and the HH band. For the first resolution level (0), only the low value is used to represent the LL band, and the high value is set to a maximum of 0, and a minimum of 15, to indicate no data is available. The minimum and maximum bit-plane table for TSSP53 is shown in the following table.

	res[0]		res[1]		...	res[n]	
	low	high	low	high	...	low	high
High Nibble	Max bitpl	0	Max	Max	...	Max	Max
Low Nibble	Min bitpl	15	Min	Min	...	Min	Min

Sorting and refinement bits table

For each resolution layer we known the number of bit planes from the minimum and maximum bit-plane table. We define data blocks for each bit plane, in each resolution layer, and specify the number of sorting and refinement bits in a special table. This table is used not only to determine the size of the different block, but also their location in the bitstream.

TSSP

For the regular TSSP, the number of blocks is defined by:

$$blocks = \sum_{n=0}^{res} BitPlRes_{max}[n] - BitPlRes_{min}[n] + 1 \tag{B.1}$$

Information for each block is then put in the stream sequentially.

nSort	nRef

Item	Size	Description
nSort	uint32	Number of bits in sorting pass
nRef	uint32	Number of bits in refinement pass

TSSP53

For the TSSP53, the number of blocks is defined by:

$$blocks = \sum_{n=0}^{res} \sum_{b=low}^{high} BitPlRes_{max}[n][b] - BitPlRes_{min}[n][b] + 1 \tag{B.2}$$

Information for each block is then put in the stream sequentially.

$nSort_{low}$	$nRef_{low}$	$nSort_{high}$	$nRef_{high}$

Item	Size	Description
$nSort_{low}$	uint32	Number of bits in low sorting pass
$nRef_{low}$	uint32	Number of bits in low refinement pass
$nSort_{high}$	uint32	Number of bits in high sorting pass
$nRef_{high}$	uint32	Number of bits in high refinement pass

B.3 Entropy data

The start position of the entropy data is located after the header, at the first alignment boundary. The alignment is specified in the header as discussed in Section B.2. Data in the stream is organized in separate blocks, one for each (bit plane, resolution, sort/ref) possibility with the start of each block properly aligned. The order of the stream is also specified in the header, as either quality progressive or resolution progressive.

Quality progressive

For a quality progressive bitstream, the block order is as follows, with R for resolution and BP for bit plane:

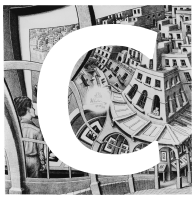
R[0], BP[max]	R[1], BP[max]	...	R[max], BP[max]
R[0], BP[max-1]	R[1], BP[max-1]	...	R[max-1], BP[max-1]
R[0], BP[max-2]	R[1], BP[max-2]	...	R[max-2], BP[max-2]

Resolution progressive

For a resolution progressive bitstream, the block order is as follows, with R for resolution and BP for bit plane:

R[0], BP[max]	R[0], BP[max-1]	...	R[0], BP[min]
R[1], BP[max]	R[1], BP[max-1]	...	R[1], BP[min]
R[2], BP[max]	R[2], BP[max-1]	...	R[2], BP[min]

APPENDIX



Rounding implications in the fixed-point implementation

In this appendix, we investigate fixed-point rounding implications that can occur with bit-plane based wavelet codecs such as EZW, SPIHT, SPECK and TSSP. We initiate this discussion with TSSP as an example, but the same issues occur in the other codecs and fixed point implementations of for example SPIHT and SPECK suffer from exactly the same behavior as TSSP due to the fact that they determine the significance of pixels in an identical manner.

C.1 Wavelet filters and two's complement numbers

The possible erroneous behavior can occur for coefficient values of the maximum negative number that can be represented in a two's complement number representation. For systems that encode wavelet coefficients, maximum negative values will not occur when the processing chain is designed properly. The growth of DC wavelet coefficients should be limited (by limiting the number of decompositions and by design of the wavelet DC scaling factors), such that this particular coefficient value will not occur. For a system using an energy-corrected $5/3$ wavelet with a factor 2 for DC coefficients, the maximum number of active bit planes after the wavelet transform is defined by:

$$BP_{max} = BP_{input} + 1 + N_{decompositions}, \tag{C.1}$$

with BP_{max} the maximum number of bit planes, BP_{input} the number of bit planes of the input and $N_{decomposition}$ the number of wavelet decompositions. The increment of unity in the equation is due to the additional bit required for encoding the sign of the wavelet result. In the case of 8-bit input values and 6 wavelet decompositions, the maximum number of bit planes is 15, so that the possible erroneous behavior will never occur. We should be aware of this behavior, and therefore the following sections discuss it in more detail, and suggest alternative solutions to optimize behavior of TSSP for this particular case.

In Section C.2, we will first discuss how the significance of numbers is determined within TSSP, what are consequences of maximum negative numbers at the decoder, and how this problem is solved for TSSP. Alternatively, the TSSP could also be modified to handle two's complement numbers differently, which is discussed in Section C.3.

C.2 Current handling of two's complement numbers

The current implementation uses $abs(value)$ to find the first important bit of a coefficient, because this makes sure the positive and negative values are treated sym-

metrically. 4 bit example:

$$\begin{aligned} 4 &= 0100, |4| = 4 = 0100, \text{first bit that is one is at bit position 2.} \\ -4 &= 1100, |-4| = 4 = 0100, \text{first bit that is one is at bit position 2.} \end{aligned}$$

For TSSP, -4 is just as significant as 4, which is represented for the whole numberspace in Figure C.1.

Figure C.1: Significance of positive and negative numbers in TSSP.

However, to support the special case of -8 = 1000, we see we need an additional bit plane to support the -1 * 8. Instead of 3 bit-plane buffers (the sign is encoded within the sorting pass buffer) we need 4 bit-plane buffers. Furthermore, in the decoder, BPR causes problems, as the -8 is coded as -1 * 8. 8 has all refinement bits 0, and does not grow like other numbers. Subtracting the last rounding correction, makes it overflow, and positive. A solution to this is using an adder/subtracter with saturation, or checking for this special case. A schematic diagram of how the values 7, -7 and -8 are created in the decoder, with and without correction with S=Sorting, R=Refinement and C=Correction is shown in Figure C.2.

C.3 Alternative handling of two's complement numbers

For the other possible handling, not the abs(value) is used, but the original two's complement value. This will avoid the extra buffer for -32768 and the special check at the decoder. 4 bit example:

$$\begin{aligned} 4 &= 0100, = \text{positive value, first bit that is one is at bit position 2.} \\ -4 &= 1100, = \text{negative value, first bit that is zero is at bit position 1.} \end{aligned}$$

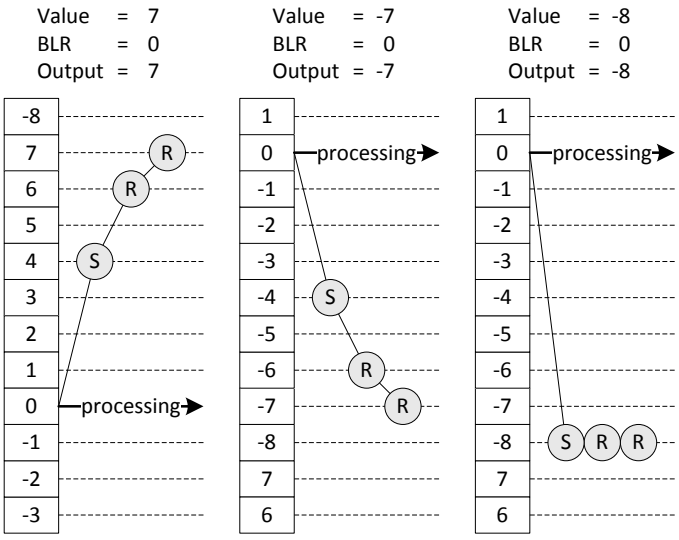


Figure C.2: Errors due to two's complement processing of a maximally negative number for a BPR of (a) 0 and (b) 1.

However, with this method, -4 and 4 are now not equal in significance. The number space is as visualized in Figure C.3.

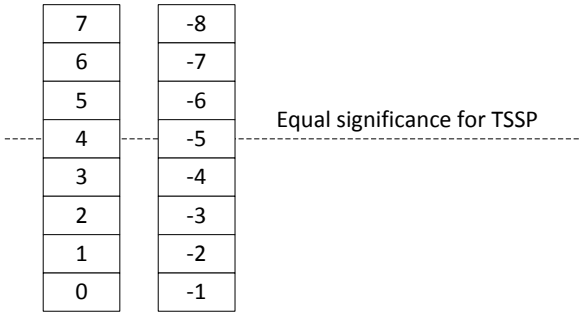


Figure C.3: Alternative significance of positive and negative numbers in TSSP.

The coefficients 4 and -5 are now equal in significance, not 4 and -4. For a BPR of 2 this means a value of 2 is kept (decoded as 3 though, due to correction) and a value of -2 is removed and decoded as 0. How this a-symmetry affect lossy coding performance is not known, but it should be fairly insignificant. Lossless coding and near lossless coding (BPR¡1) is not possible, since all -1 values have the same significance as the value 0, and are therefore treated as insignificant by the encoder.

The maxlevel should also be implemented differently. In the current implementation, the abs(value) is used, and bits are checked determining the location of the first bit that is one. This should be replaced by a routine that separates negative and positive values, and determines the location of the first one for positive values, and the location of the first zero for negative values.

Another option is to invert negative values bitwise before determining the first bit that is one. This can be done without if-statements like this: $value \oplus (value \gg 16)$, where $\oplus$ represents the binary XOR operator. 4 bit example:

$$4 = 0100, \gg 4 = 0000, 0100 \oplus 0000 = 0100, \text{first bit of one is at bit position 2.}$$
$$-4 = 1100, \gg 4 = 1111, 1100 \oplus 1111 = 0011, \text{first bit of one is at bit position 1.}$$

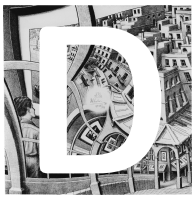
C.4 Handling of maximum negative values in TSSP

For `int16` wavelet coefficients, we usually require only 15 sorting and 15 refinement buffers, as the sign is stored in the sorting buffer of the top bit plane. To support the special case of the maximum negative number an integer can contain, we opted to utilize an additional bit-plane buffer to support maximum negative

values, as to retain symmetry in the importance of wavelet coefficients. Furthermore, by using this method of calculating the significance of wavelet coefficients, TSSP stays backward compatible with SPECK, and also the rate-distortion performance remains unaltered.

It should be noted that the special case discussed in this appendix, is not only applicable to TSSP, but also to other well-known bit-plane based coders, such as EZW, SPIHT and SPECK. Furthermore, when using natural images and standard wavelet transforms, it is impossible that this extreme negative value occurs when the number of wavelet decompositions is limited as discussed previously.

APPENDIX



Tables for calculation of arbitrary access

This appendix contains tables that depict the number of required frames for decoding an arbitrary frame. For each of the frames in the GOP, the number of frames required for decoding is given, and the average is calculated. Results are given for a predictive codec with: (1) IP structure, (2) IBP structure and (3) IBBP structure, and (4) a 3D-subband codec, in Tables D.1, D.2, D.3 and D.4, respectively.

Table D.1: Number of frames required to decode a specific arbitrary frame in a predictive IP coder with a GOP of 16.

GOP index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I-frames	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
P-frames	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B-frames	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Req. frames	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Average	**8.5 frames**															

Table D.2: Number of frames required to decode a specific arbitrary frame in a predictive IBP coder with a GOP of 16.

GOP index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I-frames	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
P-frames	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7	7
B-frames	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Req. frames	1	3	2	4	3	5	4	6	5	7	6	8	7	9	8	10
Average	**5.5 frames**															

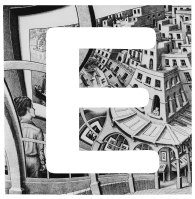
Table D.3: Number of frames required to decode a specific arbitrary frame in a predictive IBBP coder with a GOP of 15.

GOP index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I-frames	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
P-frames	0	1	1	1	2	2	2	3	3	3	4	4	4	4	4
B-frames	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1
Req. frames	1	3	3	2	4	4	3	5	5	4	6	6	5	7	7
Average	**4.33 frames**														

Table D.4: Number of frames required to decode a specific arbitrary frame in a 4-level 3D subband coder with a GOP of 16, for various temporal configurations.

GOP index	(4,0)	(1,2)	(0,2)	(0,1)	(0,0)	BDLD
0	9	5	3	2	1	1
1	14	9	6	4	2	6
2	13	8	5	3	2	5
3	14	9	6	5	3	6
4	12	7	4	3	2	4
5	14	9	7	5	3	6
6	13	8	6	4	3	5
7	14	9	7	6	4	6
8	11	6	4	3	2	3
9	14	10	7	5	3	6
10	13	9	6	4	3	5
11	14	10	7	6	4	6
12	12	8	5	4	3	4
13	14	10	8	6	4	6
14	13	9	7	5	4	5
15	14	10	8	7	5	6
Average	**13**	**8.5**	**6**	**4.5**	**3**	**5**

APPENDIX



Utilized raw test videos

In this appendix, we present the raw test videos utilized in this thesis. The test videos were chosen for the following four reasons:

- First, the test videos are available in raw YUV format, and therefore are not contaminated with any coding artifacts.
- Second, by utilizing common test videos available in the public domain, the test results can be easily compared to other literature.
- Third, sequences have been chosen that can represent typical video surveillance applications, for example a camera rotating around a scene of interest, or tracking an object. None of the scenes have scene-cuts or other editing applied to them, as this does not occur for surveillance type videos.
- Fourth, a variety of common camera resolutions have been chosen to represent variations and developments in video surveillance, from standard definition 4CIF (704×576 pixels) to high definition videos at 720p (1280×720 pixels) and 1080p (1920×1080 pixels).

Table E.1 lists the chosen videos, and provides detailed information for each regarding color space, resolution, aspect ratio, frame rate and length. The following sections will give a short description of each of the videos, and present a visual overview by showing thumbnails at varying moments in time.

Table E.1: Details of the raw video used for experimentation.

Video	Color space / Color sub-sampling	Resolution [pixels]	Aspect Ratio	Frame rate [fps]	Length [frames]
City	YC_bC_r / 4:2:0	704×576	4:3	60	600
Crew	YC_bC_r / 4:2:0	704×576	4:3	60	600
Parkrun	YC_bC_r / 4:2:0	1280×720	16:9	50	504
Stockholm	YC_bC_r / 4:2:0	1280×720	16:9	59.94	604
Station 2	YC_bC_r / 4:2:0	1920×1080	16:9	25	313
Crowd run	YC_bC_r / 4:2:0	1920×1080	16:9	50	500

City (704×576 pixels, 60 fps)
View from helicopter circling around Empire State Building, New York. Highly detailed video with complex camera movement.


Frame 0


Frame 250


Frame 500

Crew (704×576 pixels, 60 fps)
Camera following space-shuttle crew walking towards camera, with bright photography flashes. Camera starts static, then pans to right.


Frame 0


Frame 250


Frame 500

Parkrun (1280×720 pixels, 50 fps)
Camera following man running in the park, with highly detailed and random background. Camera pans from left to right.


Frame 150


Frame 350

Stockholm (1280×720 pixels, 59.94 fps)
High viewpoint on Stockholm, with highly detailed structured objects. Camera pans over image.


Frame 200


Frame 400

Station 2 (1920×1080 pixels, 25 fps)
Zoom out of train tracks, with moving train. Scene contains many lines vanishing in the horizon and hard edges.

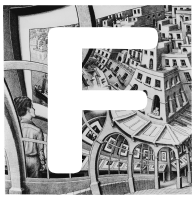

Frame 100


Frame 200

Crowd run (1920×1080 pixels, 50 fps)
Overview scene of crowd running towards camera. Camera has overview position and performs a slow pan.


Frame 100


Frame 300

APPENDIX 

Example of optimized code for the DM642 DSP

In this appendix, a code example is provided for the optimization of the 5/3 integer wavelet presented in Section 6.3. The provided code in Listing F.1 performs the predict step of the 5/3 wavelet, fully utilizing SIMD instructions and 64-bit read and writes, with the *prolog*, *kernel* and *epilog* stages clearly defined.

```
// Prolog
a_lo = _lo( _amemd8( pLineIn ) );
a_hi = _hi( _amemd8( pLineIn ) );

// Kernel
for (x = 4; x < width; x += 4)
{
  // Predict
  b_lo = _lo( _amemd8( &pLineIn[x] ) );
  b_hi = _hi( _amemd8( &pLineIn[x] ) );

  x0 = _pack2( a_hi, a_lo );
  x1 = _pack2( b_lo, a_hi );
  x2 = _shr2( _add2( x0, x1 ), 1 );
  x3 = _sub2( _packh2(a_hi, a_lo), x2 );

  y_hi = _packh2( x3, x0 );
  y_lo = _pack2( x3, x0 );

  _amemd8( &pLineIn[x - 4] ) = _itod( y_hi, y_lo );

  a_lo = b_lo;
  a_hi = b_hi;
}

// Epilog
x0 = _pack2( b_hi, b_lo );
x1 = _pack2( b_hi, b_hi );
x2 = _shr2( _add2( x0, x1 ), 1 );
x3 = _sub2( _packh2(b_hi, b_lo), x2 );

y_hi = _packh2( x3, x0 );
y_lo = _pack2( x3, x0 );

_amemd8( &pLineIn[width - 4] ) = _itod( y_hi, y_lo );
```

Listing F.1: Predict step of the 5/3 integer wavelet

Bibliography

[1] The statistics of CCTV. *BBC News*, 20 Jul 2009. Accessed: 16 Nov 2014. [Online]. Available: http://news.bbc.co.uk/2/hi/uk/8159141.stm 2

[2] You're being watched: there's one CCTV camera for every 32 people in uk. *The Guardian*, 2 Mar 2011. Accessed: 16 Nov 2014. [Online]. Available: http://www.theguardian.com/uk/2011/mar/02/cctv-cameras-watching-surveillance 2

[3] R. G. J. Wijnhoven, "Object categorization and detection and their application in surveillance," Ph.D. dissertation, Technische Universiteit Eindhoven, Nov. 2013. 4

[4] R. Albers, "Modeling and control of image processing for interventional X-ray," Ph.D. dissertation, Technische Universiteit Eindhoven, Dec. 2010. 7

[5] M. J. H. Loomans, C. J. Koeleman, K. M. Joosen, and P. H. N. de With, "Low-complexity wavelet-based image/video coding for home-use and remote surveillance," in *IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, U.S.A., Jan. 2011, pp. 641–642. 15

[6] M. J. H. Loomans and P. H. N. de With, "Complexity reduction of wavelet codecs through modified quality control," in *IEEE International Conference on Image Processing (ICIP)*, Melbourne, Australia, Sep. 2013, pp. 1670–1674. 15

[7] M. J. H. Loomans, C. J. Koeleman, and P. H. N. De With, "Low-complexity wavelet-based scalable image video coding for home-use surveillance," *Consumer Electronics, IEEE Tran. on*, vol. 57, no. 2, pp. 507–515, May 2011. 15, 215

[8] M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Scalable video compression for embedded systems in surveillance," in *Symposium on Information Theory in the Benelux*, Enschede, The Netherlands, May 2007, pp. 157–164. 15

[9] ——, "Temporal signal energy correction and low-complexity encoder feedback for lossy scalable video coding," in *IEEE Picture Coding Symposium (PCS)*, Nagoya, Japan, Dec. 2010, pp. 394–397. 15

[10] ——, "Highly-parallelized motion estimation for scalable video coding," in *IEEE International Conference on Image Processing (ICIP)*, Cairo, Egypt, Nov. 2009, pp. 1577–1580. 15

[11] ——, "Enhanced prediction for motion estimation in scalable video coding," in *IEEE International Conference on Image Processing (ICIP)*, Hong Kong, Sep. 2010, pp. 1301–1304. 15

[12] ——, "Real-time multi-level wavelet lifting scheme on a fixed-point dsp for jpeg 2000 and scalable video coding," in *IEEE International Conference on Digital Signal Processing (DSP)*, Santorini, Greece, Jul. 2009, pp. 1–6. 16

[13] ——, "Real-time two-stage speck (tssp) design and implementation for scalable video coding on embedded systems," in *IEEE Visual Communications and Image Processing (VCIP)*, Tainan, Taiwan, Nov. 2011, pp. 1–4. 16

[14] ——, "Real-time scalable video codec implementation for surveillance," in *IEEE International Conference on Multimedia and Expo (ICME)*, New York, U.S.A., Jun. 2009, pp. 1130–1133. 16

[15] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, Jul. 2003. 21, 130

[16] P. Burt and E. Adelson, "The laplacian pyramid as a compact image code," *Communications, IEEE Transactions on*, vol. 31, no. 4, pp. 532–540, Apr. 1983. 23, 91

[17] S. G. Mallat, *Multiresolution approximation and wavelets*. Department of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, Sep. 1987. 23

[18] ——, "A theory for multiresolution signal decomposition: the wavelet representation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 11, no. 7, pp. 674–693, Jul. 1989. 23

[19] ——, "Multiresolution approximations and wavelet orthonormal bases of $l^2(r)$," *Transactions of the American Mathematical Society*, vol. 315, no. 1, pp. 69–87, Sep. 1989. 23

[20] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Communications on Pure and Applied Mathematics*, vol. 41, no. 7, pp. 909–996, Oct. 1988. 23

[21] I. Daubechies *et al.*, *Ten lectures on wavelets*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, Jun. 1992. 23, 93

[22] A. Calderbank, I. Daubechies, W. Sweldens, and Boon-Lock Yeo, "Lossless image compression using integer to integer wavelet transforms," in *Image Processing, 1997. Proceedings., International Conference on*, vol. 1, Oct. 1997, pp. 596–599. 23, 93, 170

[23] ——, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332 – 369, Jul. 1998. 23, 93, 170

[24] D. Le Gall and A. Tabatabai, "Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques," in *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, vol. 2, Apr. 1988, pp. 761–764. 23, 93, 170

[25] A. Cohen, I. Daubechies, and J.-C. Feauveau, "Biorthogonal bases of compactly supported wavelets," *Communications on Pure and Applied Mathematics*, vol. 45, no. 5, pp. 485–560, Jun. 1992. 23, 93, 170

[26] A. Said and W. Pearlman, "An image multiresolution representation for lossless and lossy compression," *Image Processing, IEEE Transactions on*, vol. 5, no. 9, pp. 1303–1310, Sep. 1996. 23

[27] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *Image Processing, IEEE Transactions on*, vol. 1, no. 2, pp. 205–220, Apr. 1992. 23

[28] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186 – 200, Apr. 1996. 23, 24, 88, 171

[29] ——, "The lifting scheme: A construction of second generation wavelets," *SIAM Journal on Mathematical Analysis*, vol. 29, no. 2, pp. 511–546, Mar. 1998. 23

[30] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 247–269, May 1998. 23

[31] Zhang Li-bao and Wang Ke, "Embedded multiple subbands scaling image coding using reversible integer wavelet transforms," in *Intelligent Multimedia, Video and Speech Processing, 2004. Proceedings of 2004 International Symposium on*, Apr. 2004, pp. 599–602. 26, 170, 215

[32] A. S. Lewis and G. Knowles, "A 64 Kb/s video codec using the 2-D wavelet transform," in *Data Compression Conference, 1991. DCC '91.*, Apr. 1991, pp. 196–201. 28

[33] J. Shapiro, "An embedded wavelet hierarchical image coder," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 4, Mar. 1992, pp. 657–660. 28

[34] ——, "An embedded hierarchical image coder using zerotrees of wavelet coefficients," in *Data Compression Conference, 1993. DCC '93.*, Mar. 1993, pp. 214–223. 28

[35] ——, "Embedded image coding using zerotrees of wavelet coefficients," *Signal Processing, IEEE Transactions on*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993. 28

[36] M. Rabbani and P. W. Jones, *Digital image compression techniques*. SPIE Press, Feb. 1991, vol. 7. 28

[37] J. Shapiro, "A fast technique for identifying zerotrees in the EZW algorithm," in *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, vol. 3, May 1996, pp. 1455–1458. 28

[38] C. D. Creusere, "Spatially partitioned lossless image compression in an embedded framework," in *Signals, Systems amp; Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*, vol. 2, Oct. 1997, pp. 1455–1459. 29

[39] ——, "A new method of robust image compression based on the embedded zerotree wavelet algorithm," *Image Processing, IEEE Transactions on*, vol. 6, no. 10, pp. 1436–1442, Oct. 1997. 29

[40] A. Said and W. Pearlman, "Image compression using the spatial-orientation tree," in *Circuits and Systems, 1993., ISCAS '93, 1993 IEEE International Symposium on*, May 1993, pp. 279–282. 29

[41] ——, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 6, no. 3, pp. 243–250, Jun. 1996. 29, 30

[42] G. Sullivan and R. Baker, "Efficient quadtree coding of images and video," in *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, vol. 4, Apr. 1991, pp. 2661–2664. 29

[43] ——, "Efficient quadtree coding of images and video," *Image Processing, IEEE Transactions on*, vol. 3, no. 3, pp. 327–331, May 1994. 29

[44] A. Islam and W. A. Pearlman, "Embedded and efficient low-complexity hierarchical image coder," *Visual Communications and Image Processing 1999*, vol. 3653, no. 1, pp. 294–305, Jan. 1999. 29, 39

[45] W. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, low-complexity image coding with a set-partitioning embedded block coder," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 14, no. 11, pp. 1219–1235, Oct. 2004. 29, 39

[46] F. W. Wheeler and W. Pearlman, "Combined spatial and subband block coding of images," in *Image Processing, 2000. Proceedings. 2000 International Conference on*, vol. 3, Sep. 2000, pp. 861–864. 29

[47] D. Taubman, "High performance scalable image compression with EBCOT," in *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, vol. 3, Oct. 1999, pp. 344–348. 29

[48] ——, "High performance scalable image compression with EBCOT," *Image Processing, IEEE Transactions on*, vol. 9, no. 7, pp. 1158–1170, Jul. 2000. 29, 53

[49] M. Marcellin, M. Gormish, A. Bilgin, and M. Boliek, "An overview of JPEG-2000," in *Data Compression Conference, 2000. Proceedings. DCC 2000*, Mar. 2000, pp. 523–541. 29

[50] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: an overview," *Consumer Electronics, IEEE Transactions on*, vol. 46, no. 4, pp. 1103–1127, Oct. 2000. 29

[51] F. W. Wheeler and W. Pearlman, "Low-memory packetized SPIHT image compression," in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, vol. 2, Oct. 1999, pp. 1193–1197. 29

[52] S. J. C. Geerts, D. Cornelissen, and P. H. N. de With, "Embedded image enhancement for high-throughput cameras," in *Video Surveillance and Transportation Imaging Applications 2014*, vol. 9026. SPIE, Mar. 2014, pp. 90 260Z–90 260Z–14. 29

[53] D. W. J. M. van de Wouw, K. van Rens, H. van Lint, E. G. T. Jaspers, and P. H. N. de With, "Real-time change detection for countering improvised explosive devices," in *Video Surveillance and Transportation Imaging Applications 2014*, vol. 9026. SPIE, Mar. 2014, pp. 90 260T–90 260T–15. 29

[54] R. DeVore, B. Jawerth, and B. Lucier, "Image compression through wavelet transform coding," *Information Theory, IEEE Transactions on*, vol. 38, no. 2, pp. 719–746, Mar. 1992. 30

[55] Wen-Kuo Lin and N. Burgess, "Listless zerotree coding for color images," in *Signals, Systems amp; Computers, 1998. Conference Record of the Thirty-Second Asilomar Conference on*, vol. 1, Oct. 1998, pp. 231–235. 52

[56] Wen-Kuo Lin, Brian Wai-Him Ng, N. Burgess, and A. Bouzerdoum, "Reduced memory zerotree coding algorithm for hardware implementation," in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 2, Jul. 1999, pp. 57–61. 52

[57] Wen-Kuo Lin, A. Moini, and N. Burgess, "Listless zerotree coding using raster tree search," in *TENCON 2001. Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology*, vol. 2, Aug. 2001, pp. 514–518. 52

[58] F. W. Wheeler and W. Pearlman, "SPIHT image compression without lists," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, vol. 4, Jun. 2000, pp. 2047–2050. 52

[59] P. Schelkens, F. Decroos, G. Lafruit, F. Catthoor, and J. Cornelis, "Efficient implementation of embedded zero-tree wavelet encoding," in *Electronics, Circuits and Systems, 1999. Proceedings of ICECS '99. The 6th IEEE International Conference on*, vol. 2, Sep. 1999, pp. 1155–1158. 52

[60] Jiangling Guo, S. Mitra, B. Nutter, and T. Karp, "An efficient image codec based on backward coding of wavelet trees," in *Image Analysis and Interpretation, 2006 IEEE Southwest Symposium on*, Mar. 2006, pp. 233–237. 52

[61] ——, "A fast and low complexity image codec based on backward coding of wavelet trees," in *Data Compression Conference, 2006. DCC 2006. Proceedings*, Mar. 2006, pp. 292–301. 52

[62] C. D. Creusere, "Fast embedded compression for video," *Image Processing, IEEE Transactions on*, vol. 8, no. 12, pp. 1811–1816, Dec. 1999. 53

[63] J.-R. Ohm, "Three-dimensional subband coding with motion compensation," *Image Processing, IEEE Transactions on*, vol. 3, no. 5, pp. 559–571, Sep. 1994. 88, 92

[64] Seung-Jong Choi and J. Woods, "Motion-compensated 3-D subband coding of video," *Image Processing, IEEE Transactions on*, vol. 8, no. 2, pp. 155–167, Feb. 1999. 88, 92, 117

[65] A. Secker and D. Taubman, "Motion-compensated highly scalable video compression using an adaptive 3D wavelet transform based on lifting," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, vol. 2, Oct. 2001, pp. 1029–1032. 88, 92

[66] B. Pesquet-Popescu and V. Bottreau, "Three-dimensional lifting schemes for motion compensated video compression," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, vol. 3, May 2001, pp. 1793–1796. 88, 92

[67] M. van der Schaar and D. Turaga, "Unconstrained motion compensated temporal filtering (umctf) framework for wavelet video coding," in *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, vol. 2, Jul. 2003, pp. II–581–4. 88, 92, 99

[68] D. Turaga and M. van der Schaar, "Reduced complexity spatio-temporal scalable motion compensated wavelet video encoding," in *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, vol. 2, Jul. 2003, pp. II–561–4. 88

[69] D. Turaga, M. van der Schaar, and B. Pesquet-Popescu, "Complexity scalable motion compensated wavelet video encoding," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 15, no. 8, pp. 982–993, Aug. 2005. 88

[70] G. Pau, B. Pesquet-Popescu, M. van der Schaar, J. Vieron, and J. Viron, "Delay-performance trade-offs in motion-compensated scalable subband video compression," in *Proc. of Advanced Concepts for Intelligent Vision Systems (ACIVS)*, Sep. 2004. 88

[71] G. Pau, J. Vieron, and B. Pesquet-Popescu, "Video coding with flexible MCTF structures for low end-to-end delay," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 3, Sep. 2005, pp. III–241–4. 88, 100, 101

[72] T. Sikora, "MPEG digital video-coding standards," *Signal Processing Magazine, IEEE*, vol. 14, no. 5, pp. 82–100, Sep. 1997. 90

[73] B. G. Haskell, *Digital video: an introduction to MPEG-2*. Springer, Dec. 1996. 90

[74] J.-R. Ohm, "Temporal domain sub-band video coding with motion compensation," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 3, Mar. 1992, pp. 229–232. 92

[75] D. Taubman and A. Zakhor, "Multirate 3-D subband coding of video," *Image Processing, IEEE Transactions on*, vol. 3, no. 5, pp. 572–588, Sep. 1994. 92

[76] A. Secker and D. Taubman, "Highly scalable video compression using a lifting-based 3D wavelet transform with deformable mesh motion compensation," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 3, Sep. 2002, pp. 749–752. 92

[77] ——, "Lifting-based invertible motion adaptive transform (limat) framework for highly scalable video compression," *Image Processing, IEEE Transactions on*, vol. 12, no. 12, pp. 1530–1542, Dec. 2003. 92

[78] Beong-Jo Kim and W. Pearlman, "An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (spiht)," in *Data Compression Conference, 1997. DCC '97. Proceedings*, Mar. 1997, pp. 251–260. 92

[79] Beong-Jo Kim, Zixiang Xiong, and W. Pearlman, "Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-d spiht)," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 10, no. 8, pp. 1374–1387, Dec. 2000. 92

[80] Shih-Ta Hsiang and J. W. Woods, "Embedded video coding using invertible motion compensated 3-D subband/wavelet filter bank," *Signal Processing: Image Communication*, vol. 16, no. 8, pp. 705 – 724, Jul. 2001, packet Video Communications. 92

[81] Shih-Ta Hsiang, J. Woods, and J.-R. Ohm, "Invertible temporal subband/wavelet filter banks with half-pixel-accurate motion compensation," *Image Processing, IEEE Transactions on*, vol. 13, no. 8, pp. 1018–1028, Aug. 2004. 92

[82] Peisong Chen and J. Woods, "Bidirectional mc-ezbc with lifting implementation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 14, no. 10, pp. 1183–1194, Oct. 2004. 92, 130

[83] Yongjun Wu and J. Woods, "Scalable motion vector coding based on cabac for mc-ezbc," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 6, pp. 790–795, Jun. 2007. 92

[84] Yongjun Wu, K. Hanke, T. Rusert, and J. Woods, "Enhanced mc-ezbc scalable video coder," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 10, pp. 1432–1436, Oct. 2008. 92

[85] ——, "Corrections to enhanced mc-ezbc scalable video coder," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 3, pp. 452–452, Mar. 2009. 92

[86] V. Bottreau, M. Benetiere, B. Felts, and B. Pesquet-Popescu, "A fully scalable 3D subband video codec," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, vol. 2, Oct. 2001, pp. 1017–1020. 92

[87] Y. Andreopoulos, A. Munteanu, J. Barbarien, M. V. der Schaar, J. Cornelis, and P. Schelkens, "In-band motion compensated temporal filtering," *Signal Processing: Image Communication*, vol. 19, no. 7, pp. 653 – 673, Aug. 2004, special Issue on Subband/Wavelet Interframe Video Coding. 92

[88] G. Pau and B. Pesquet-Popescu, "Uniform motion-compensated 5/3 filterbank for subband video coding," in *Image Processing, 2004. ICIP '04. 2004 International Conference on*, vol. 5, Oct. 2004, pp. 3117–3120. 92

[89] C. Tillier, B. Pesquet-Popescu, and M. van der Schaar, "Weighted average spatio-temporal update operator for subband video coding," in *Image Processing, 2004. ICIP '04. 2004 International Conference on*, vol. 2, Oct. 2004, pp. 1305–1308. 92

[90] ——, "Improved update operators for lifting-based motion-compensated temporal filtering," *Signal Processing Letters, IEEE*, vol. 12, no. 2, pp. 146–149, Feb. 2005. 92, 100

[91] ——, "3-band motion-compensated temporal structures for scalable video coding," *Image Processing, IEEE Transactions on*, vol. 15, no. 9, pp. 2545–2557, Sep. 2006. 92

[92] M. Trocan, C. Tillier, B. Pesquet-Popescu, and M. van der Schaar, "A 5-band temporal lifting scheme for video surveillance," in *Multimedia Signal Processing, 2006 IEEE 8th Workshop on*, Oct. 2006, pp. 278–281. 92

[93] Y. Andreopoulos, A. Munteanu, G. Van der Auwera, P. Schelkens, and J. Cornelis, "A new method for complete-to-overcomplete discrete wavelet transforms," in *Digital Signal Processing, 2002. DSP 2002. 2002 14th International Conference on*, vol. 2, Jul. 2002, pp. 501–504. 93

[94] ——, "Scalable wavelet video-coding with in-band prediction - implementation and experimental results," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 3, Jun. 2002, pp. 729–732. 93

[95] Y. Andreopoulos, M. van der Schaar, A. Munteanu, J. Barbarien, P. Schelkens, and J. Cornelis, "Fully-scalable wavelet video coding using in-band motion compensated temporal filtering," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, vol. 3, Apr. 2003, pp. III–417–20. 93

[96] Y. Andreopoulos, A. Munteanu, G. Van der Auwera, J. P. H. Cornelis, and P. Schelkens, "Complete-to-overcomplete discrete wavelet transforms: theory and applications," *Signal Processing, IEEE Transactions on*, vol. 53, no. 4, pp. 1398–1412, Apr. 2005. 93

[97] K. Hanke, J.-R. Ohm, and T. Rusert, "Adaptation of filters and quantization in spatio-temporal wavelet coding with motion compensation," in *Proc. of the Picture Coding Symposium*, Apr. 2003, pp. 49–54. 100

[98] Jiangtao Wen and J. Villasenor, "Reversible variable length codes for efficient and robust image and video coding," in *Data Compression Conference, 1998. DCC '98. Proceedings*, Apr. 1998, pp. 471–480. 117

[99] Kai-Kuang Ma and Gang Qiu, "Unequal-arm adaptive rood pattern search for fast block-matching motion estimation in the JVT/H.26L," in *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, vol. 1, Sep. 2003, pp. I–901–4. 117, 130, 132, 134

[100] Renxiang Li, Bing Zeng, and Ming L. Liou, "A new three-step search algorithm for block motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 4, no. 4, pp. 438–442, Apr. 1994. 130

[101] A. M. Tourapis, O. C. L. Au, and Ming L. Liou, "Predictive motion vector field adaptive search technique (PMVFAST): enhancing block-based motion estimation," in *in the Optimization Model 1.0, in ISO/IEC JTC1/SC29/WG11 MPEG2000/M6194*, Jan. 2001, pp. 883–892. 131, 132, 135, 140

[102] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in *Proceedings of Visual Communications and Image Processing*, vol. 4671, Jan. 2002, pp. 1069–1079. 131, 132, 135, 140

[103] Weiyao Lin, K. Panusopone, D. Baylon, and Ming-Ting Sun, "A new class-based early termination method for fast motion estimation in video coding," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, Sep. 2009, pp. 625–628. 131

[104] G. de Haan, P. Biezen, H. Huijgen, and O. Ojo, "True-motion estimation with 3-D recursive search block matching," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 3, no. 5, pp. 368–379, 388, Oct. 1993. 133, 140

[105] Yao Nie and Kai-Kuang Ma, "Adaptive rood pattern search for fast block-matching motion estimation," *Image Processing, IEEE Transactions on*, vol. 11, no. 12, pp. 1442–1449, Dec. 2002. 134

[106] Shan Zhu and Kai-Kuang Ma, "A new diamond search algorithm for fast block-matching motion estimation," *Image Processing, IEEE Transactions on*, vol. 9, no. 2, pp. 287–290, Aug. 2000. 134

[107] L. P. J. Vosters, "Efficient enhancement and extraction of depth for 3-D video," Ph.D. dissertation, Technische Universiteit Eindhoven, Oct. 2014. 136

[108] J. A. Vijverberg, M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Two novel motion-based algorithms for surveillance video analysis on embedded platforms," in *Real-Time Image and Video Processing (RTIVP)*. Brussels, Belgium: SPIE, May 2010, p. 77240I. 148

[109] M. D. Adams and F. Kossentini, "Performance evaluation of reversible integer-to-integer wavelet transforms for image compression," in *Proc. IEEE Data Compression Conference*. Citeseer, Mar. 1999, p. 514. 170

[110] M. Adams and F. Kossentni, "Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis," *Image Processing, IEEE Transactions on*, vol. 9, no. 6, pp. 1010–1024, Jun. 2000. 170

[111] S. Chatterjee and C. Brooks, "Cache-efficient wavelet lifting in JPEG 2000," in *Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference on*, vol. 1, Aug. 2002, pp. 797–800. 170

[112] P. Meerwald, R. Norcen, and A. Uhl, "Cache issues with JPEG2000 wavelet lifting," *Visual Communications and Image Processing 2002*, vol. 4671, no. 1, pp. 626–634, Jan. 2002. 170

[113] Byeong-Doo Choi, Kang-Sun Choi, Min-Cheol Hwang, Jun-ki Cho, and Sung-Jea Ko, "Real-time DSP implementation of motion-JPEG2000 using overlapped block transferring and parallel-pass methods," *Real-Time Imaging*, vol. 10, no. 5, pp. 277 – 284, Oct. 2004. 170, 197

Publication List

Journal papers

[1] I. M. Creusen, S. Javanbakhti, M. J. H. Loomans, L. Hazelhoff, N. Roubtsova, S. Zinger, and P. H. N. de With, "Vicomo: Visual context modelling for scene understanding in video surveillance," *Journal of Electronic Imaging*, vol. 22, no. 4, pp. 041 117–1–1/19, Jan. 2013.

[2] M. J. H. Loomans, C. J. Koeleman, and P. H. N. De With, "Low-complexity wavelet-based scalable image video coding for home-use surveillance," *Consumer Electronics, IEEE Tran. on*, vol. 57, no. 2, pp. 507–515, May 2011.

Winner first-price Best Paper Award

[3] M. J. H. Loomans, C. J. Koeleman, K. M. Joosen, and P. H. N. de With, "Low-complexity wavelet-based image/video coding for home-use and remote surveillance," in *IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, U.S.A., Jan. 2011, pp. 641–642.

Conference papers

[4] M. J. H. Loomans and P. H. N. de With, "Complexity reduction of wavelet codecs through modified quality control," in *IEEE International Conference on Image Processing (ICIP)*, Melbourne, Australia, Sep. 2013, pp. 1670–1674.

[5] M. J. H. Loomans, R. Wijnhoven, and P. H. N. de With, "Robust automatic ship tracking in harbours using active cameras," in *IEEE Int. Conf. on Image Processing (ICIP)*, Melbourne, Australia, Sep. 2013, pp. 4117–4121.

[6] M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Real-time two-stage speck (tssp) design and implementation for scalable video coding on embedded systems," in *IEEE Visual Communications and Image Processing (VCIP)*, Tainan, Taiwan, Nov. 2011, pp. 1–4.

[7] M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Temporal signal energy correction and low-complexity encoder feedback for lossy scalable video coding," in *IEEE Picture Coding Symposium (PCS)*, Nagoya, Japan, Dec. 2010, pp. 394–397.

[8] ——, "Enhanced prediction for motion estimation in scalable video coding," in *IEEE International Conference on Image Processing (ICIP)*, Hong Kong, Sep. 2010, pp. 1301–1304.

[9] J. A. Vijverberg, M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Two novel motion-based algorithms for surveillance video analysis on embedded platforms," in *Real-Time Image and Video Processing (RTIVP)*. Brussels, Belgium: SPIE, May 2010, p. 77240I.

[10] M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Highly-parallelized motion estimation for scalable video coding," in *IEEE International Conference on Image Processing (ICIP)*, Cairo, Egypt, Nov. 2009, pp. 1577–1580.

[11] J. A. Vijverberg, M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Global illumination compensation for background subtraction using gaussian-based background difference modeling," in *IEEE Advanced Video and Signal Based Surveillance (AVSS)*, Genoa, Italy, Sep. 2009, pp. 448–453.

[12] M. J. H. Loomans, C. J. Koeleman, and P. H. N. de With, "Real-time multi-level wavelet lifting scheme on a fixed-point dsp for jpeg 2000 and scalable video coding," in *IEEE International Conference on Digital Signal Processing (DSP)*, Jul. 2009, pp. 1–6.

[13] ——, "Real-time scalable video codec implementation for surveillance," in *IEEE International Conference on Multimedia and Expo (ICME)*, New York, U.S.A., Jun. 2009, pp. 1130–1133.

[14] ——, "Performance vs. complexity in scalable video coding for embedded surveillance applications," in *Visual Communications and Image Processing (VCIP)*. San Jose, U.S.A.: SPIE, Jan. 2008, pp. 1–11.

[15] ——, "Scalable video compression for embedded systems in surveillance," in *Symposium on Information Theory in the Benelux*, Enschede, The Netherlands, May 2007, pp. 157–164.

Acknowledgements

Writing these acknowledgements finally gives me the opportunity to answer that single question that has haunted me for many years: when are you going to graduate? While I love to travel, my PhD has been a journey like no other. Now that my journey has been put to paper, I finally have time to look back instead of ahead. With this new perspective, I now realize the great unknown lying ahead of me as a fresh PhD candidate was actually much greater than I initially anticipated. Not only in how much work is involved, but also the breath of skills involved in successfully completing an academic work of this magnitude. Reflecting back, it also shows how much it has transformed me, both personally and professionally, something one doesn't realize when looking forward. During this journey, I encountered many hurdles, but each of them have taught me something, and for each of them I came out stronger than when I went in. While a large part of this had to be accomplished in solitude, it would not have been possible without the continuous support of my family, friends and colleagues, to whom I would like to express my gratitude.

While the words on these pages are insufficient to fully cover my gratitude, I would like to thank my wife Shikin and our two sweet cats Mojo† and Sumi who have provided me with a safe haven to which I could always fall back to. Especially during the last few months of writing this thesis, I have terribly neglected them by working full-time, followed by late nights and weekends, commonly into the wee hours of the night. Instead of feeling hurt, they have always provided me with love and support, and someone to cuddle with in times of need.

Similar gratitude goes to my parents, Mari and Erma, which have provided unconditional support, combined with a tireless pride and believe in me, even at times when I was not sure I deserved it. My sister Tjitske and her husband Micha, thank you for supporting my quest and keeping our family close and in contact, even though we all live hundreds of kilometers apart. And let's not forget my godson Bennett, that one day he may improve on this work. To my uncle Ton†, thank you for always being positive and interested in my studies. Even though

you cannot be here to witness my accomplishment, I'm sure you are proud of me. Special thanks go to my family on the other side of the world. Jaafar, Maimunah, Sham, Su, Nikkisha and Natra, thank you for embracing me into your family and giving me a second home in Singapore.

Special thanks go to my promoter, prof. Peter de With for offering me the opportunity to perform this research as a student of his Video Coding and Architecture (VCA) group at the TU/e. Peter, we first met at Philips Electronics in Singapore, where you took me under your wings and introduced me to the world of video processing. Ever since, you have guided me along the path of growing into an independent researcher, while always managing to provide support in times of need. Expressing my gratitude to Peter must be extended to his wife Jelly and his son Roeland as well. Thank you for allowing me to spend so many evenings skyping with Peter. Furthermore, I would like to thank the members of the promotion committee, for the time and effort they spent in reading this thesis and the valuable feedback that followed.

Most of this research has been performed during my time at VDG Security, and gratitude goes to Kalle van den Berg and Jan van den Berg for supporting this research. Special thanks go to Rick Koeleman for greatly supporting me during the development of this work. I enjoyed our many discussions, not only with regards to the research, but also about our shared interest in music and high-quality audio. Julien, thanks for all the discussions we had and the help that you have given me over the years. Fedde, Herman and Thijs, it has been an honor to partake in the weekly luncheons at the golden arches and the fierce snooker competition with such distinguished gentlemen. Gratitude goes to Ard for embedding the scalable video decoder in the DIVA video surveillance software platform.

I also had the pleasure to be a guest researcher at ViNotion for the WaterVisie project. First of all, I would like to thank my copromotor Egbert Jaspers for giving me a second home at ViNotion. You have a great group of people around you, and I really enjoyed my time at ViNotion. Rob, Rickert and Tim, thank you for teaching me TopCode™, and thanks to the rest of the ViNotion gang for creating such a pleasant and fun working environment.

During my research, I visited the TU/e weekly, which brings me to my esteemed colleagues of the VCA group. I enjoyed working with all of you and would like to thank some people personally. Anja, thank you for the Wednesday cookies, the support at the group and for squeezing in my appointments with Peter, I know with his agenda that is no easy feat. Luat, Goran and Rob (Albers), we had some great times during our conference visits, where we explored some fascinating parts of the world. Sveta, Ivo, Lykele, Julien, Ruud and Fons, I fondly remember our many lunch discussions regarding all topics, from playing 'Go' to

parenting advice and everything in between. Ruud and Chrysie, special thanks for lifting the veil on the tribulations of writing a thesis. Ruud, I really enjoyed our hifi audio discussions and hope to one day build the sub-20 Hz subwoofer monster we talked about. Chrysie, thanks for helping me out in getting the first decent looking version of my thesis. Of course I also have to attribute Yannick for this, if you look carefully you probably recognize a certain LaTeX template at the foundation of this thesis... Raphael, Francisco, Frits and Icy, thank you for cheering up the VCA office and 'kinecting' me to the world of 3D scanning. Egor, thanks for providing some stiff competition during karting and the best quality refreshments during the PhD parties. Prarthana, I hope you are still progressing in karate on your way to the 10th Dan. Bahman, I still have to laugh at the thought that it took me one and a half years to realize you spoke Dutch fluently. Ping, we first met at Philips Singapore, and our paths crossed again in Eindhoven, what are the odds? Xinfeng, I enjoyed working with you on the WaterVisie project and hope that the Dutch government isn't pestering you anymore with cryptic Dutch letters. Onno, good luck with the last straws of your thesis. Dirk, your thesis is a reference work for the VCA group and rightfully named the VCA bible, also due to its close resemblance in the number of pages.

I am lucky to have so many personal friends that have helped me keep my sanity during this long process. Dennis and Remko, warm thanks for all the nerdy 'hacking' stuff we pulled off together and all the fun we had at the big dance festivals. We have stayed best friends while being geographically challenged, and you guys have always been there for me when it mattered. Chris and Iertje†, thank you for the chilling sessions and making hey-can-we-count-cells-using-vision-ok-lets-give-it-a-try a reality. Frank van Heesch, I greatly enjoy our technical discussions, your musical robots and SkiVips. And thank you for actually reading this thesis. Benjamin, thanks for the legal advice and always having an exciting and funny story to share about peculiar situations you somehow always manage to end up in. Wouter, you are the only person I know that loves food like a true Singaporean, and loves to travel like a true Dutchman, so let's continue to makan and travel together. Shona, thank you for always being a good (food) party enthusiast. Credit goes to Wim, Boyd, Helena and Joram for managing my strategy, especially for the numerous student parties in Rotterdam, and let's continue this tradition in Amsterdam. Also Wim, you are next! Hatice, Damira, Steef and Maarten, thank you for compelling me to get away from the computer and hit the gym to get a great workout, followed by post-workout happy juice and steak infested cheat nights. After my absence during the last few months, I hope you will all be there to instill in me your fitness craze upon my return. No pain no gain!

Curriculum Vitae

Marijn Loomans was born in Aalst-Waalre, the Netherlands, in 1977. During his MSc. study at the Eindhoven University of Technology, the Netherlands, he performed an additional Masters study at the Design Technology Institute of the National University of Singapore. After 2 years, in 2004, he received the Master of Technological Design degree (MTD) in Embedded Systems. For his MTD thesis, he focused on noise within mainstream LCD television architectures at Philips Mainstream Flat Displays, Singapore. In 2005, he received the MSc. degree in Electrical Engineering from the Eindhoven University of Technology, the Netherlands. For his thesis, he focused on local content analysis for picture quality enhancements at Philips Mainstream Flat Displays, Singapore. In 2006, he joined the Video Coding and Architectures (VCA) research group at the Eindhoven University of Technology, the Netherlands, as a PhD student. His industrial PhD was conducted at VDG Security, Zoetermeer, the Netherlands, where he researched scalable video coding architectures based on wavelets and his research interests are image and video compression, scalable video coding, wavelet-based video coding and content-based video analysis. Besides his research in the area of scalable video coding, he also investigated automated tracking using moveable Pan-Tilt-Zoom cameras, which was used to follow ships in the port of Rotterdam for the WaterVisie project, and for tracking the movement of people in the ViCoMo project. He authored and co-authored 15 publications and in 2011 he received the first-price Best Paper Award of the International IEEE Conference on Consumer Electronics for his work on scalable image and video coders. Since 2013 he is employed at Baker Hughes Reservoir Software in Delft.